OAuth 2.0 and OpenID Connect

AARON PARECKI
@aaronpk
aaronpk.com

Micropub

W3C Recommendation 23 May 2017

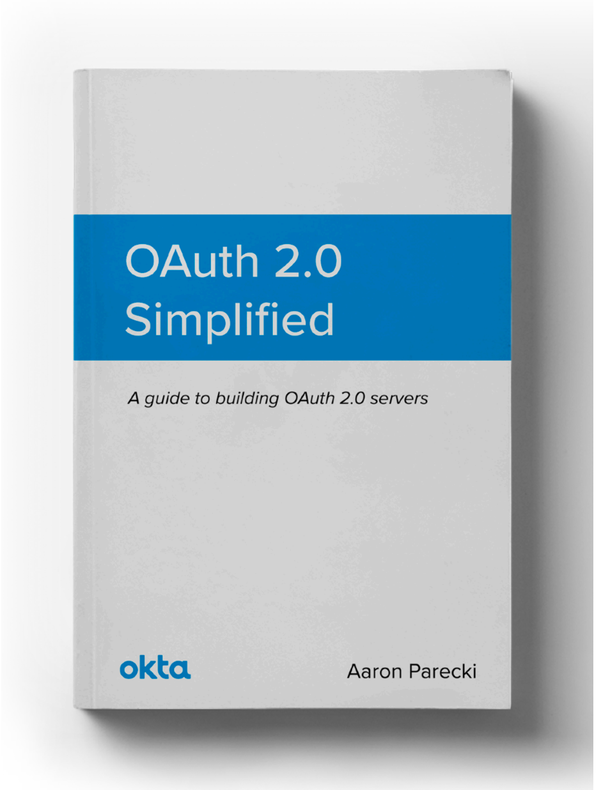
Latest published version:
https://www.w3.org/TR/micropub/

Latest editor's draft:
https://micropub.net/draft/

Previous version:
https://www.w3.org/TR/2017/PR-micropub-20170413/

Editor:
Aaron Parecki

Repository:
Github
Issues
Commits



[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-parecki-oauth-browser-based-apps)
00

Open Authentication Protocol — A. Parecki
Internet-Draft — Okta
Intended status: Best Current Practice — D. Waite
Expires: August 2, 2019 — Ping Identity
January 29, 2019

OAuth 2.0 for Browser-Based Apps
draft-ietf-oauth-browser-based-apps-00

Abstract

OAuth 2.0 authorization requests from apps running entirely in a browser are unable to use a Client Secret during the process, since they have no way to keep a secret confidential. This specification details the security considerations that must be taken into account when developing browser-based applications, as well as best practices for how they can securely implement OAuth 2.0.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

oauth.net

OAuth 2.0 Code Articles Security Books About

An **open protocol** to allow **secure authorization** in a **simple** and **standard** method from web, mobile and desktop applications.

Learn more about OAuth 2.0 »

The OAuth 2.0 authorization framework enables third-party applications to obtain limited access to a web service.

For Consumer developers...

If you're building...

- web applications
- desktop applications
- mobile applications
- Javascript or browser-based apps

OAuth is a simple way to publish and interact with protected data. It's also a safer and more secure way for people to give you access. We've kept it simple to save you time.

For Service Provider developers...

If you're supporting...

- web applications
- mobile applications
- server-side APIs
- mashups

If you're storing protected data on your users' behalf, they shouldn't be spreading their passwords around the web to get access to it. Use OAuth to give your users access to their data while protecting their account credentials.

WHAT IS OAUTH?

AND WHY DOES IT MATTER?

Updated by: 8252 PROPOSED STANDARD
Errata Exist
Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: 5849
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft

October 2012

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6749.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

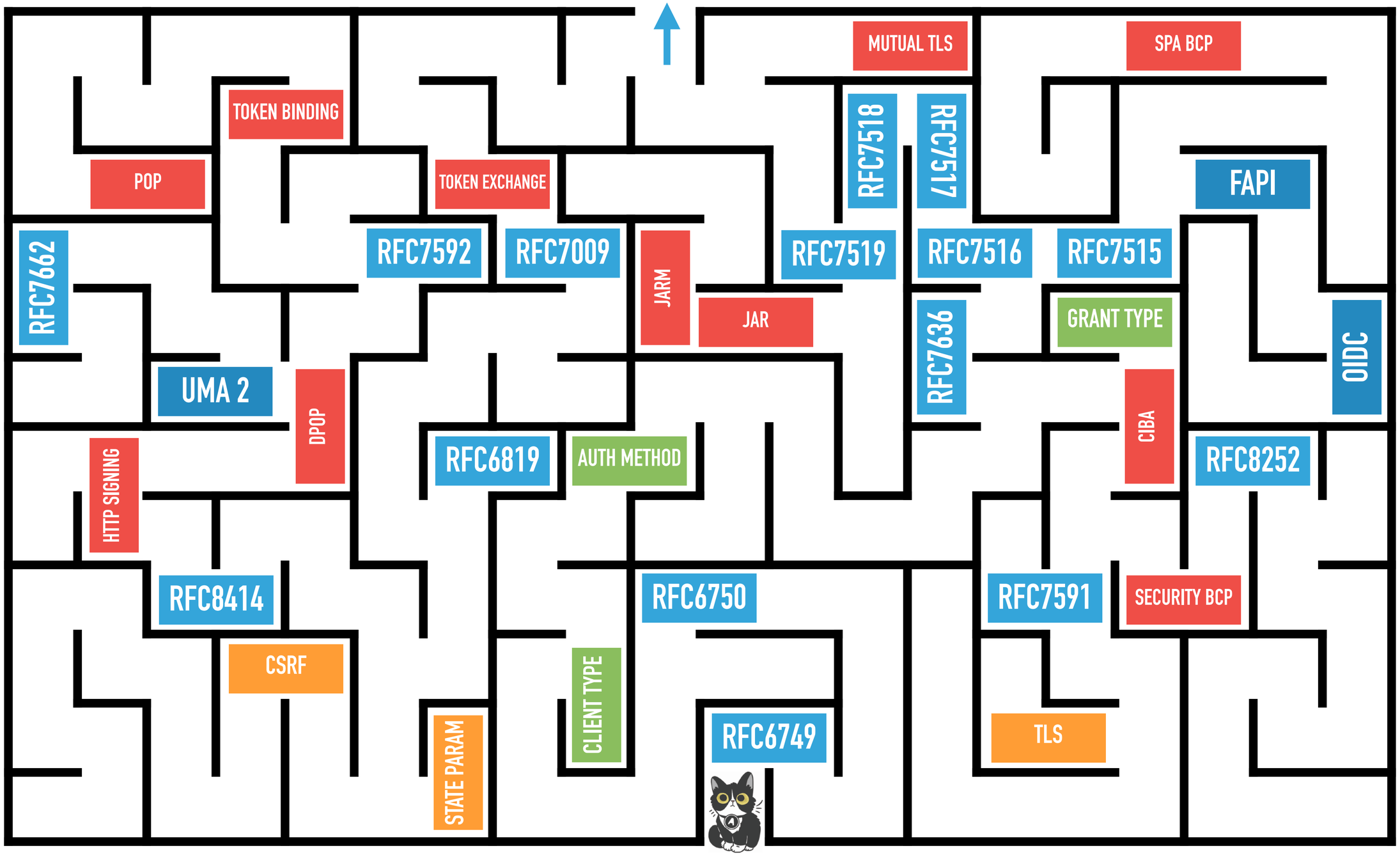
This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 4
1.1. Roles 6
1.2. Protocol Flow 7
1.3. Authorization Grant 8
1.3.1. Authorization Code 8
1.3.2. Implicit 8
1.3.3. Resource Owner Password Credentials 9
1.3.4. Client Credentials 9
1.4. Access Token 10
1.5. Refresh Token 10
1.6. TLS Version 12
1.7. HTTP Redirections 12

Specs are not good tutorials!

BUILDING YOUR APPLICATION



THE PASSWORD ANTI-PATTERN

Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service

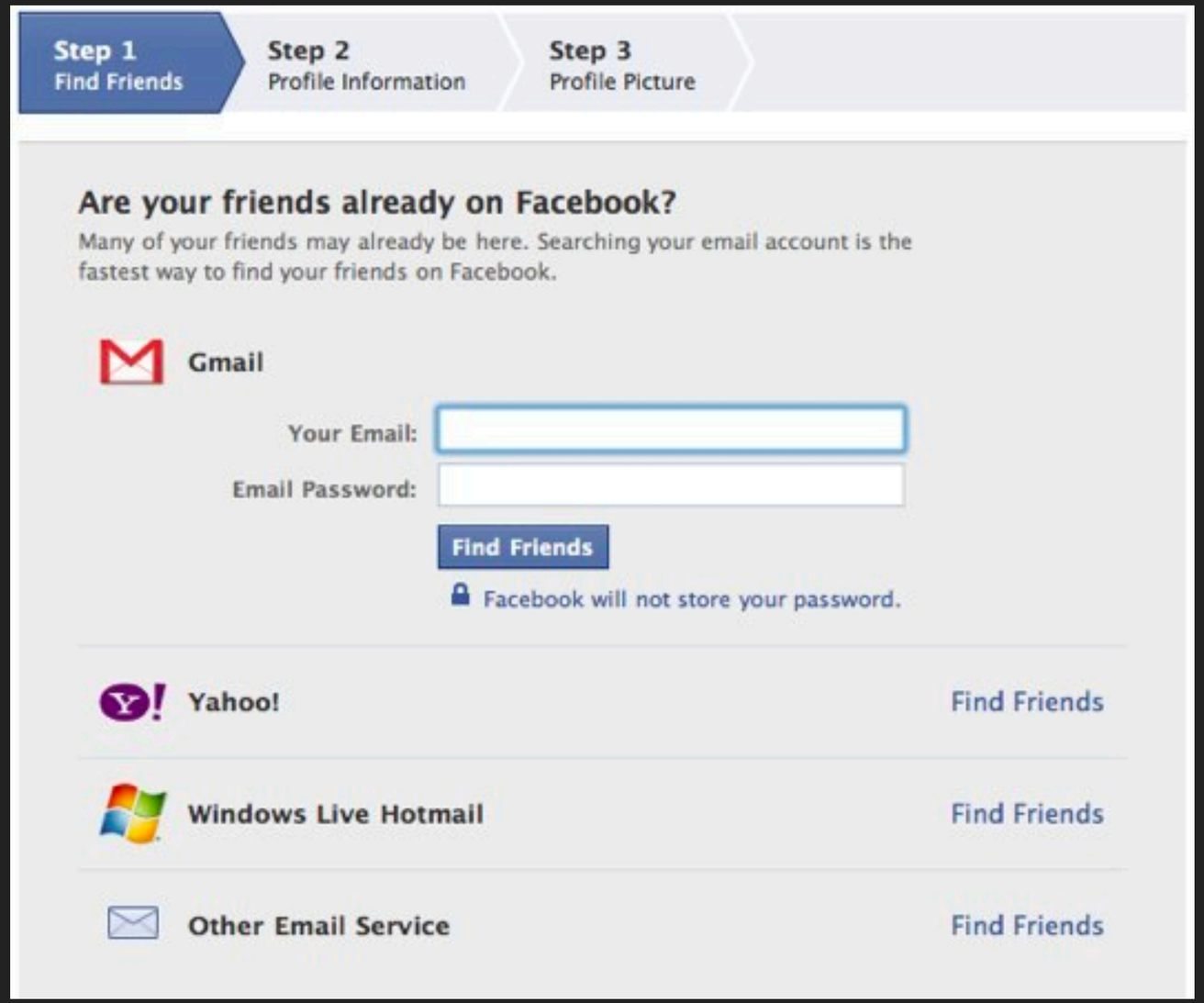
msn Hotmail | YAHOO! MAIL | AOL Mail | Gmail

Your Email Address: ima.testguy@gmail.com (e.g. bob@gmail.com)

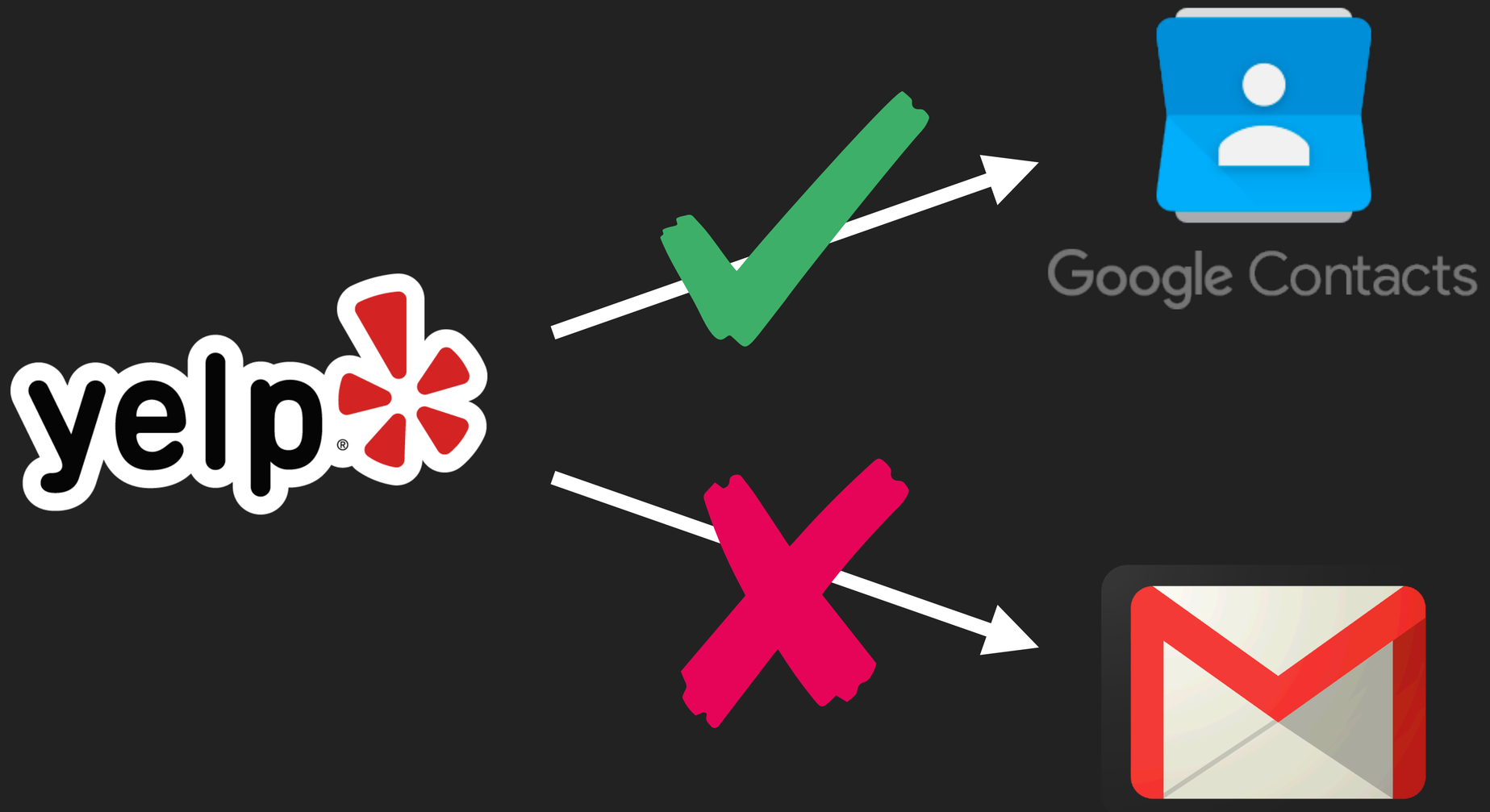
Your Gmail Password: ••••••••••• (The password you use to log into your Gmail email)

Skip this step | Check Contacts

THE PASSWORD ANTI-PATTERN



facebook.com ~2010

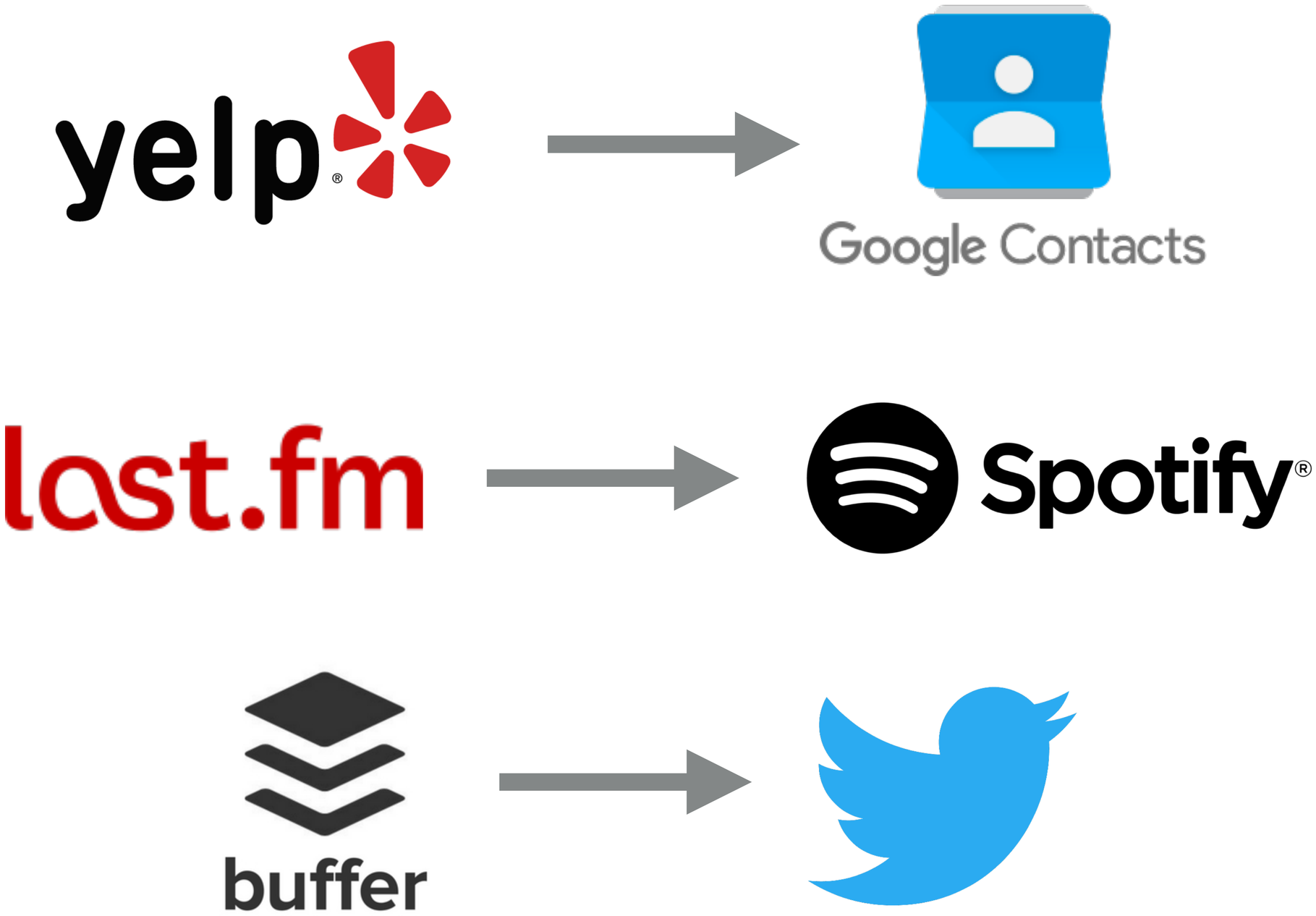

yelp
Google Contacts

so...

how can I let an app

access my data

without giving it my password?


yelp
Google Contacts
last.fm
Spotify
buffer

Google

jaiku

facebook

cliqset

magnolia

twitter

AMERICA Online

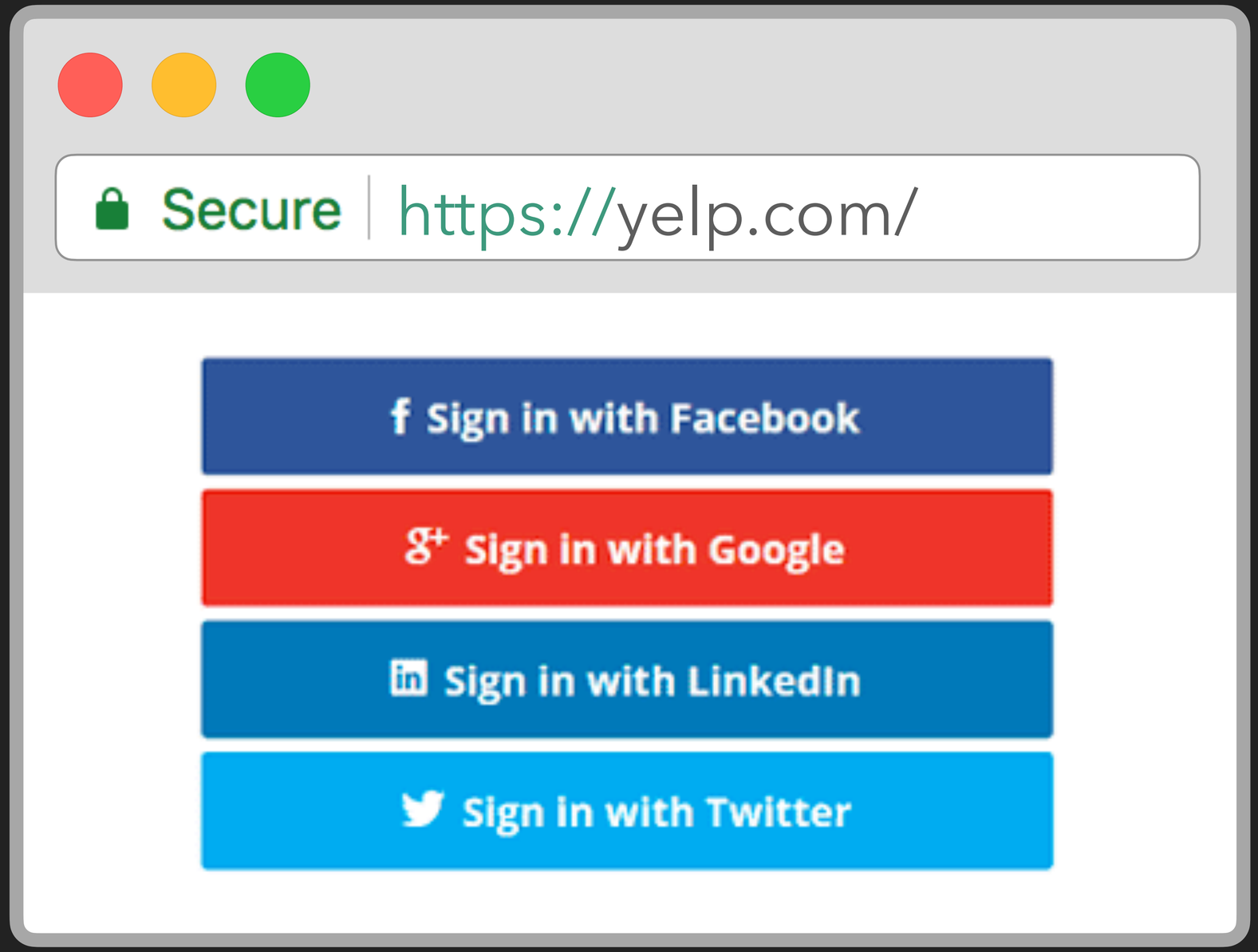
flickr

six apart

Pownce™

YAHOO!

An open standard for authorization

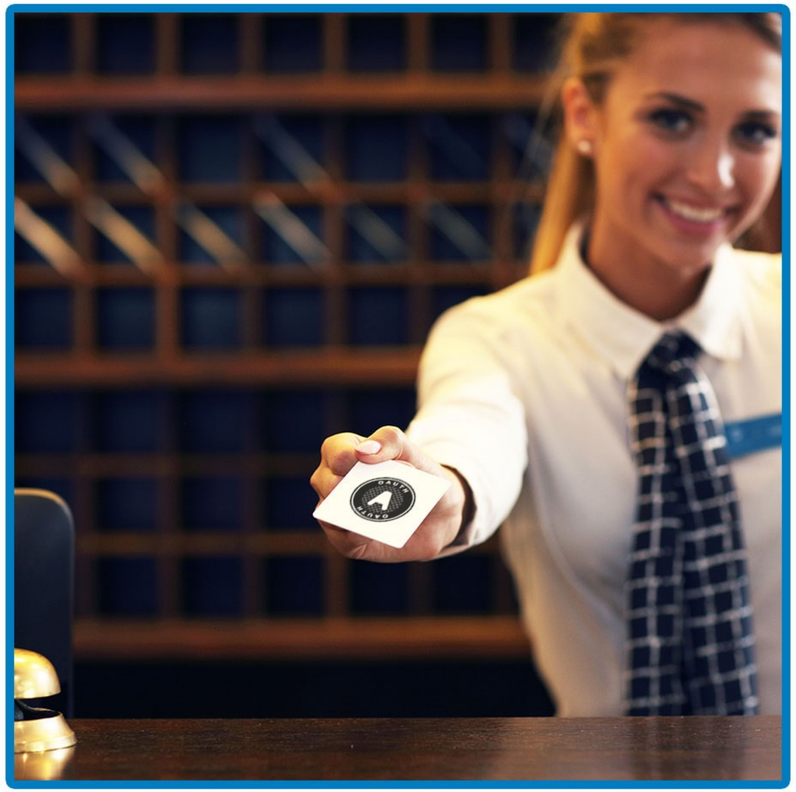

Secure
https://yelp.com/
Sign in with Facebook
Sign in with Google
Sign in with LinkedIn
Sign in with Twitter

but…

OAuth doesn’t tell the app

who

logged in



Authorization Server

Access Token

Resource (API)

Identification

OpenID Connect

authentication

Accessing APIs

OAUTH

authorization

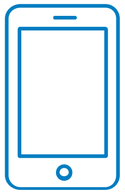
QUIZ

HOW OAUTH WORKS

OAUTH FLOWS

Authorization Code

web



mobile

SPA

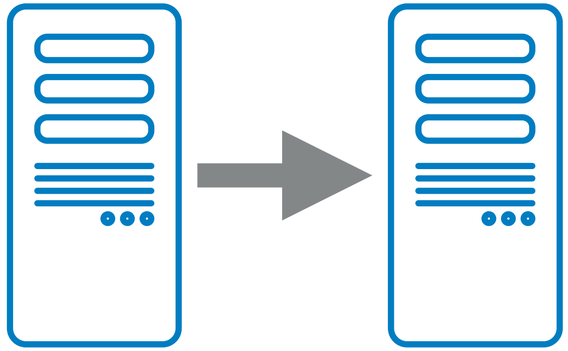
CLI

Device Flow

CLI

browserless devices

Client Credentials



server-to-server

~~Implicit~~

~~Password~~

USING AN ACCESS TOKEN

```
POST /resource/1/update HTTP/1.1
Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia
Host: api.authorization-server.com

description=Hello+World
```

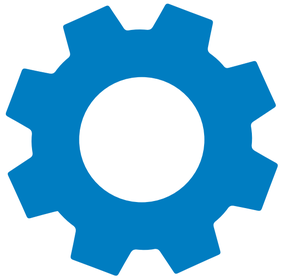
ROLES IN OAUTH

ROLES IN OAUTH

The User

Device

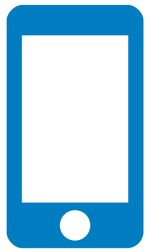
The Application



API

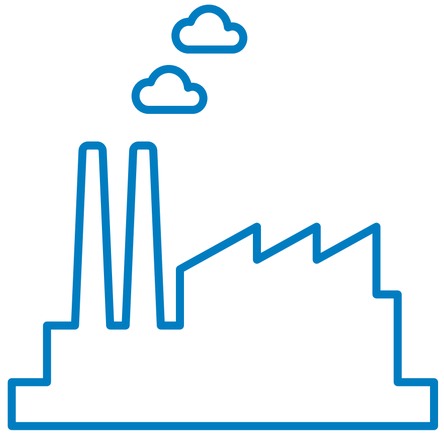
ROLES IN OAUTH

The User
(Resource Owner)

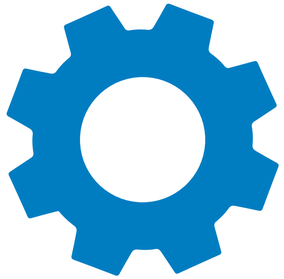


Device
(User Agent)

The Application
(Client)



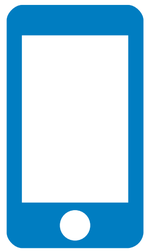
OAuth Server
(Authorization Server)
aka the token factory



API
(Resource Server)

ROLES IN OAUTH

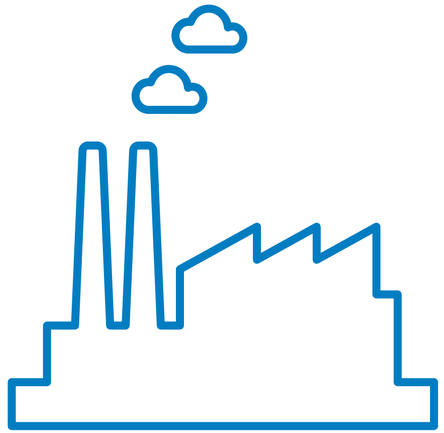
The User
(Resource Owner)



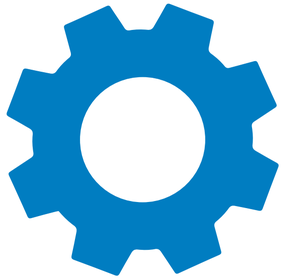
Device
(User Agent)

The Application
(Client)

Travis-CI.org



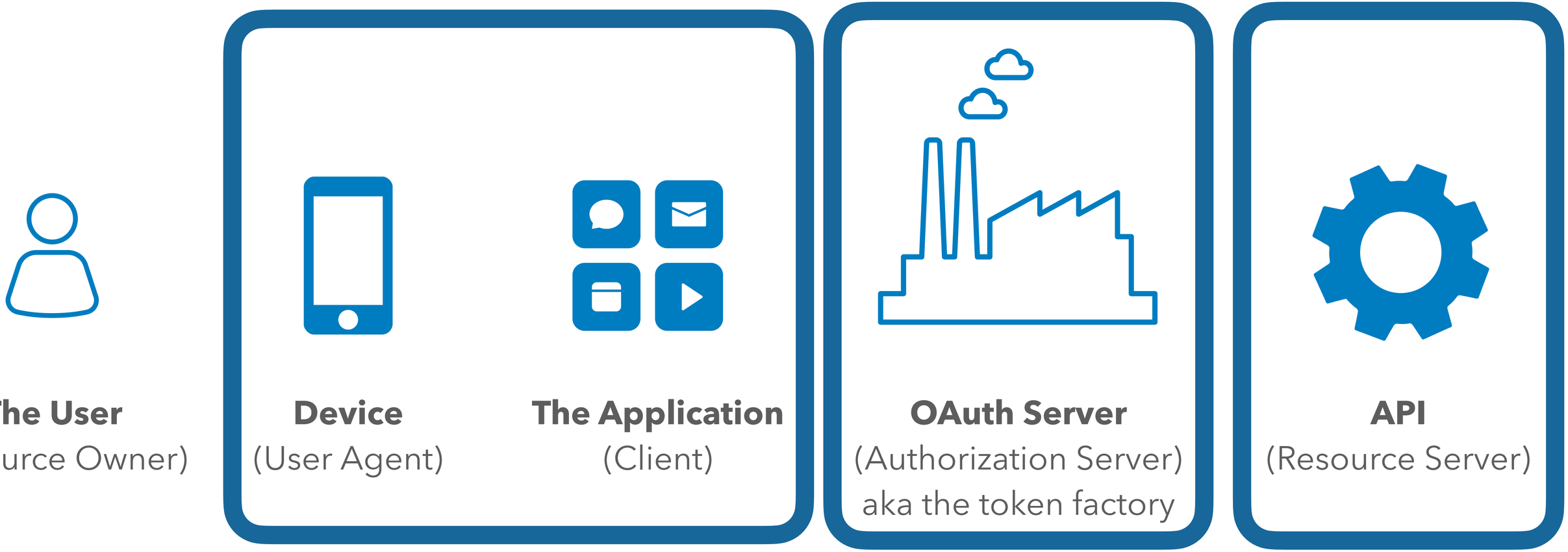
OAuth Server
(Authorization Server)
aka the token factory



API
(Resource Server)

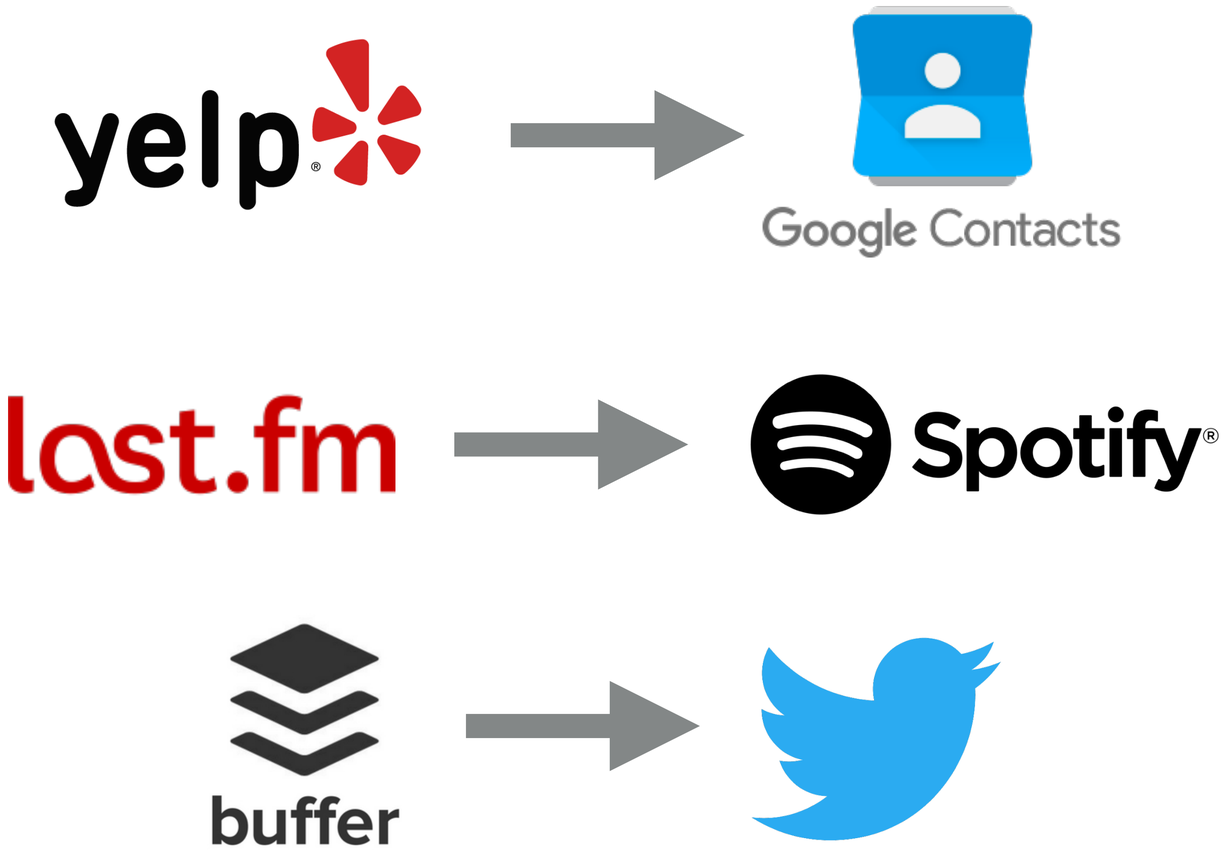
GitHub

ROLES IN OAUTH

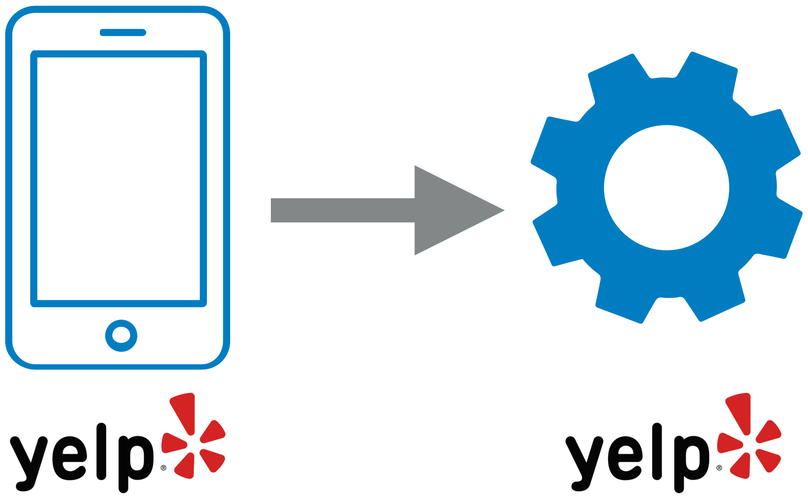


USER CONSENT

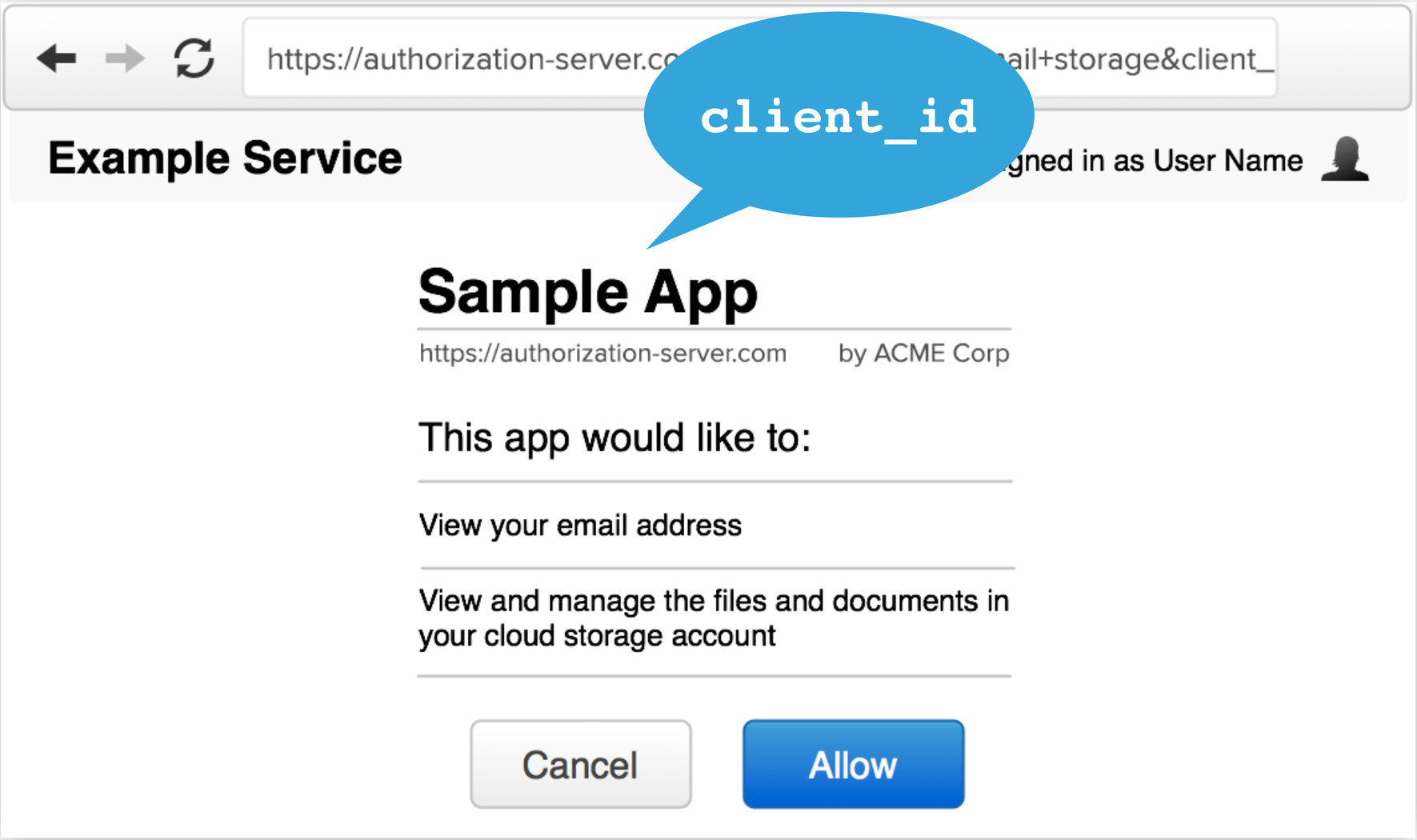
THIRD PARTY



FIRST PARTY



Authorization Interface



Authorize Example App to access your account?

Authorize app | Cancel

This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.

Learn more about third-party app permissions in the Help Center.

Example App

By Aaron Parecki

example-app.com

Free Version Control with unlimited private and public repositories.

fitbit

Example App by Aaron Parecki would like the ability to access the following data in your Fitbit account

- [x] Select All
- [x] activity and exercise
- [x] weight
- [x] sleep
- [x] food and water logs
- [x] location and GPS
- [x] profile
- [x] heart rate

Deny

Allow

Data shared with aaronpk will be governed by Aaron Parecki's privacy policy and terms of service. You can revoke this consent at any time in your Fitbit account settings. More information about these permissions can be found here.

Signed in as aaron@parecki.com
Not you?

Authorize OAuth 2 Example App

OAuth 2 Example App by **aaronpk**
wants to access your **aaronpk** account

Personal user data
Full access

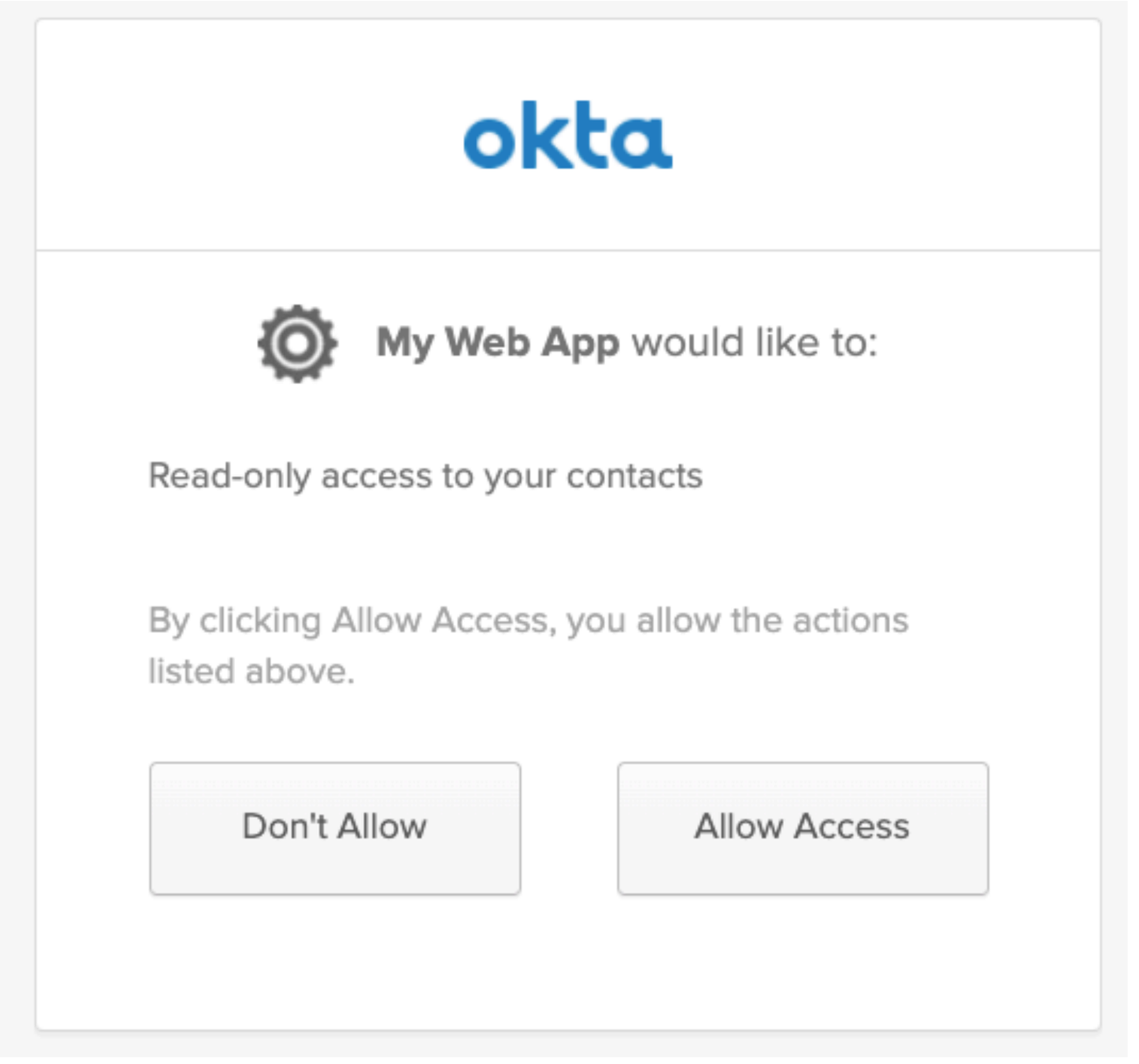
Repositories
Public only

Organization access

- indieweb ✓
- microformats ✓
- oauth2 ✓
- okta ✓
- w3c ✓

Authorize aaronpk

Authorizing will redirect to
https://example-app.com.dev

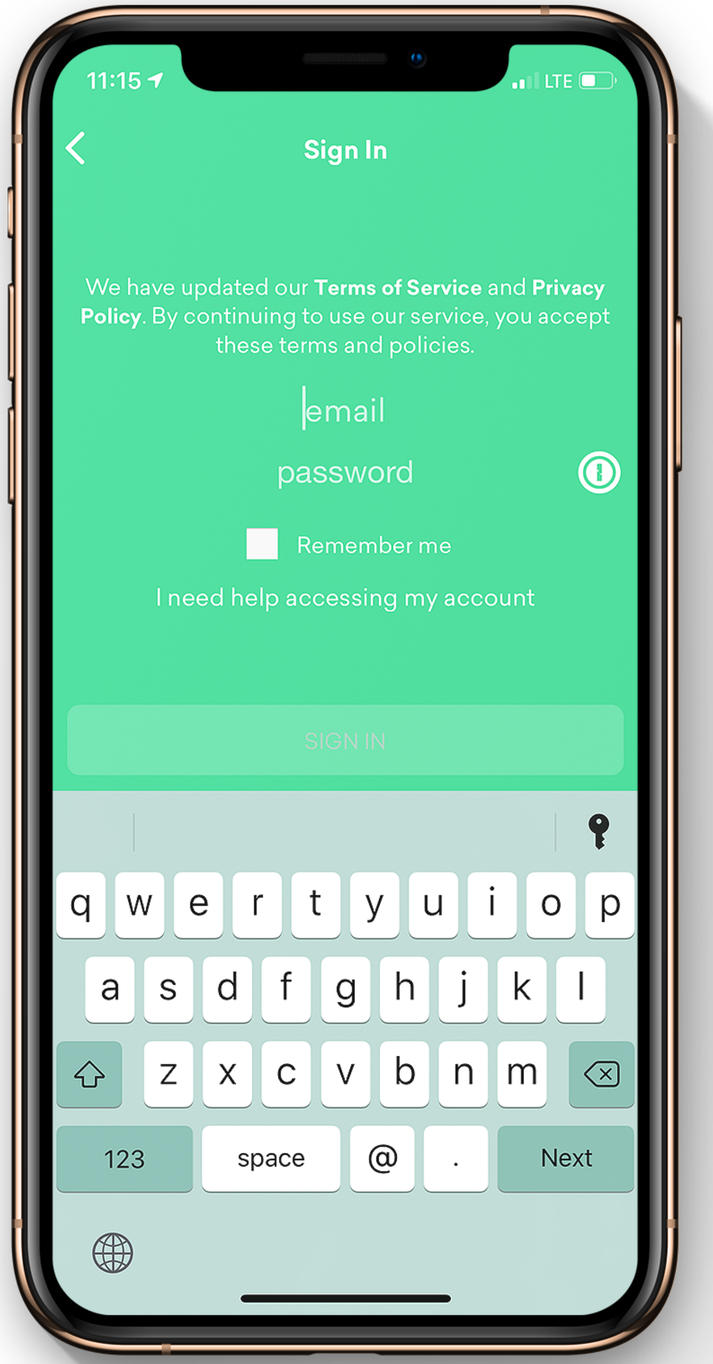

okta
My Web App would like to:
Read-only access to your contacts
By clicking Allow Access, you allow the actions listed above.
Don't Allow
Allow Access

Password Grant

```
POST https://authz-server/token
  grant_type=password&
  username=USERNAME&
  password=PASSWORD&
  client_id=CLIENT_ID
```

No client secret!

Nice looking target for testing password dumps



The Original Purpose of the Password Grant

- Added to OAuth to enable migrating applications from HTTP Basic Auth or using a stored password to OAuth

Re: [OAUTH-WG] OAuth 2.1: dropping password grant	Dick Hardt	2020-02-19

Some background for those interested. I added this grant into OAuth 2.0 to allow applications that had been provided password to migrate. Even with the caveats in OAuth 2.0, implementors decide they want to prompt the user to enter their credentials, the anti-pattern OAuth was created to eliminate.

Password Grant - Deprecated in Security BCP

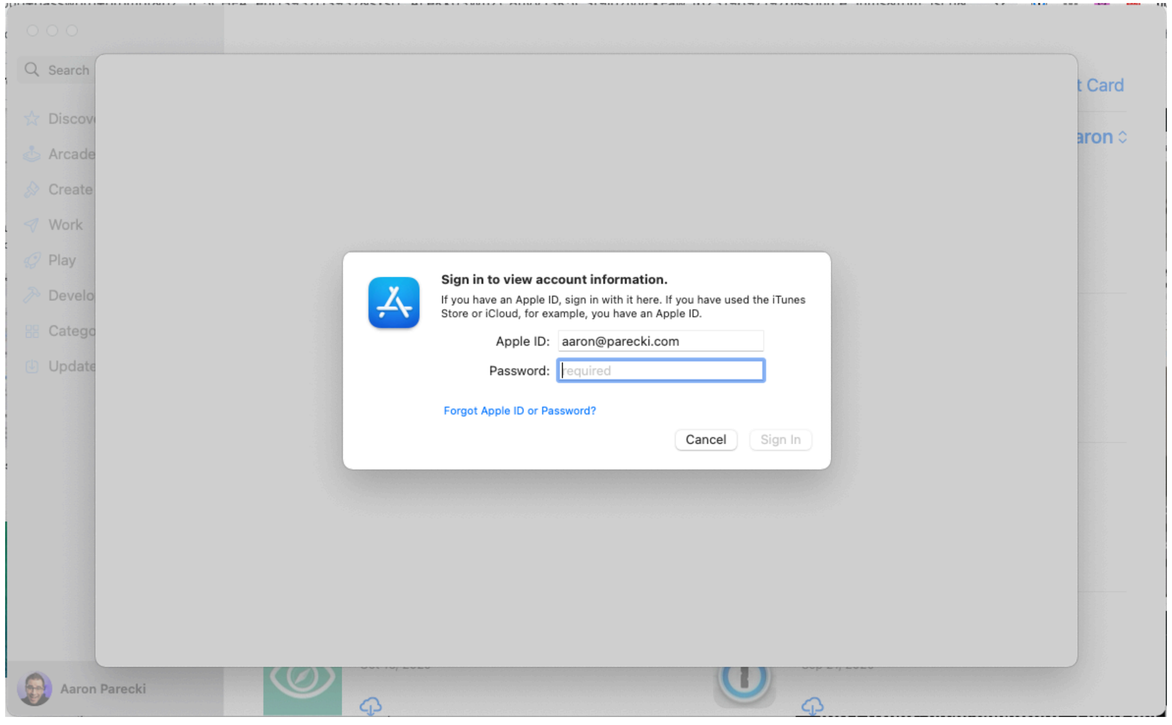
3.4. Resource Owner Password Credentials Grant

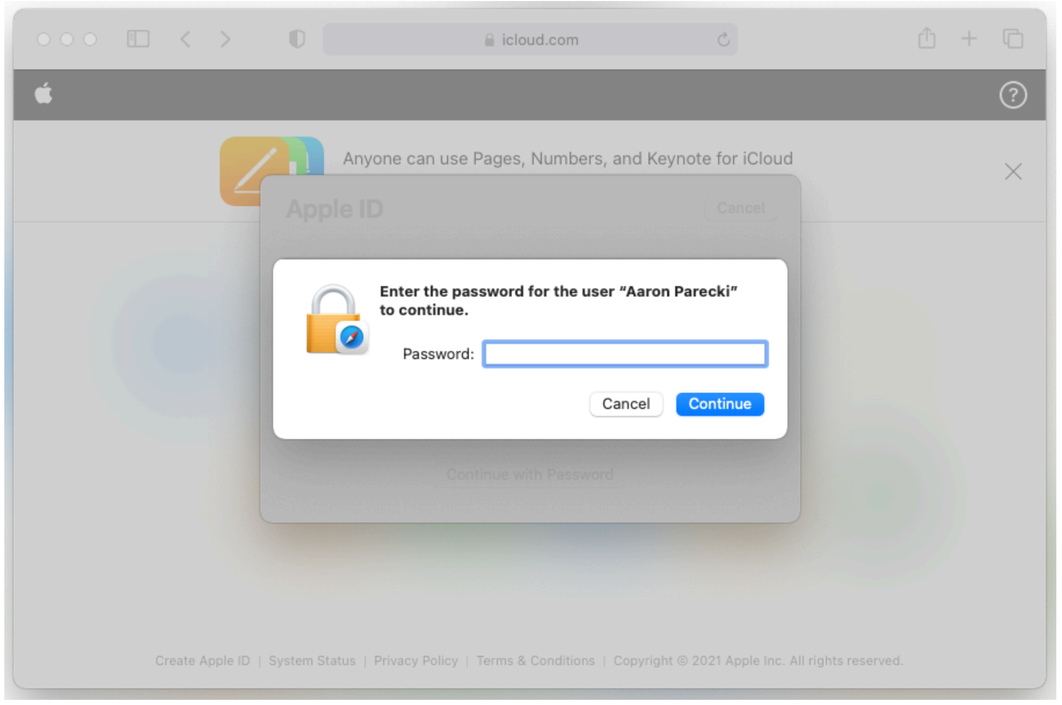
The resource owner password credentials grant MUST NOT be used. This grant type insecurely exposes the credentials of the resource owner to the client. Even if the client is benign, this results in an increased attack surface (credentials can leak in more places than just the AS) and users are trained to enter their credentials in places other than the AS.

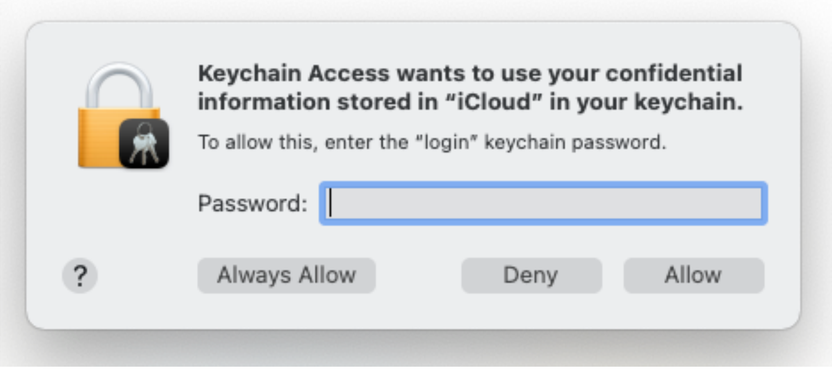
Furthermore, adapting the resource owner password credentials grant to two-factor authentication, authentication with cryptographic credentials, and authentication processes that require multiple steps can be hard or impossible (WebCrypto, WebAuthn).

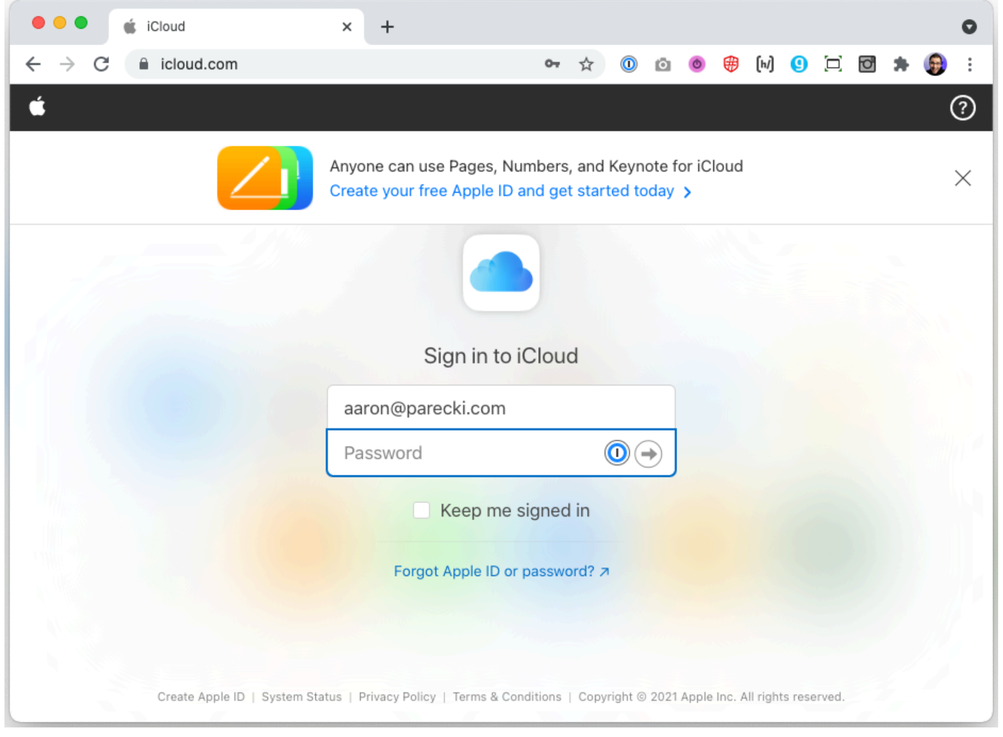
Problems with the Password Grant

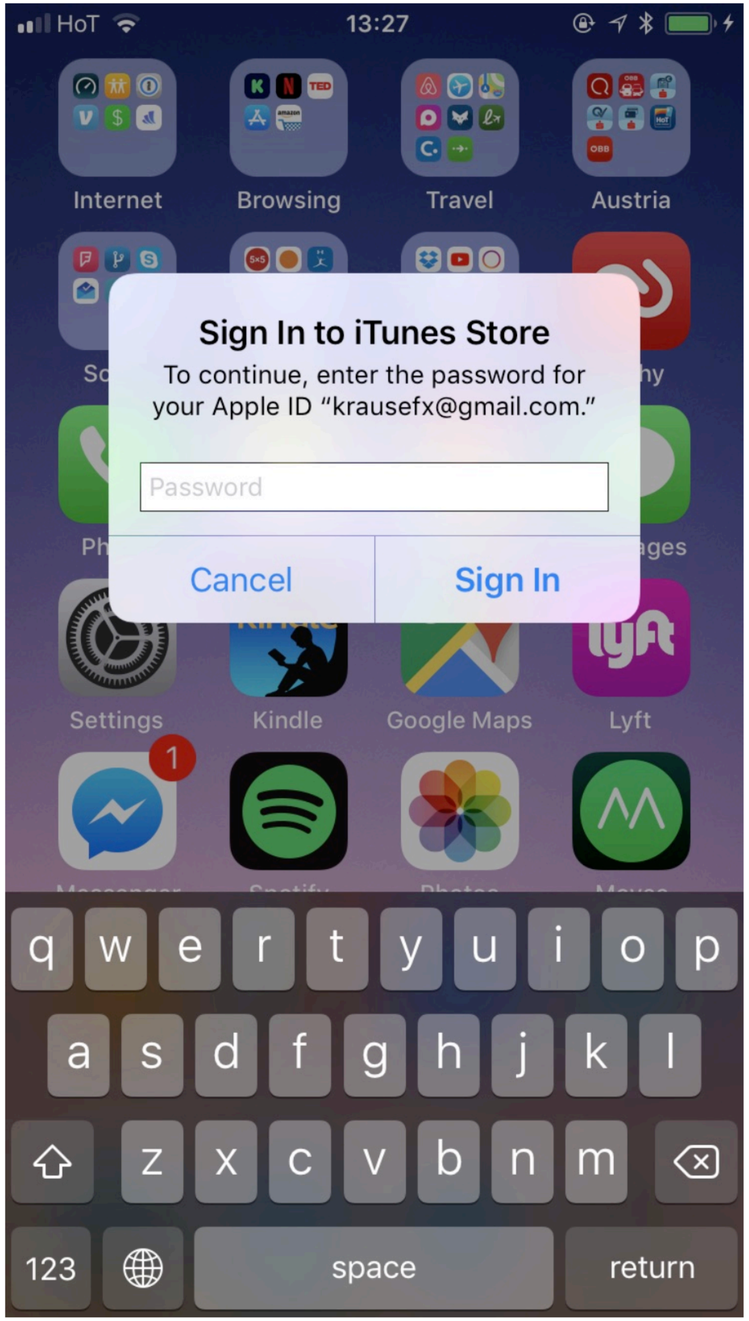
- Exposes the username and password to the application
- No way to guarantee user consent
- Even for first-party / trusted clients, this increases the attack surface
- Trains users that it's okay to enter their password in more than one place
- Difficult or impossible to extend to support multifactor or passwordless authentication (WebCrypto, WebAuthn)

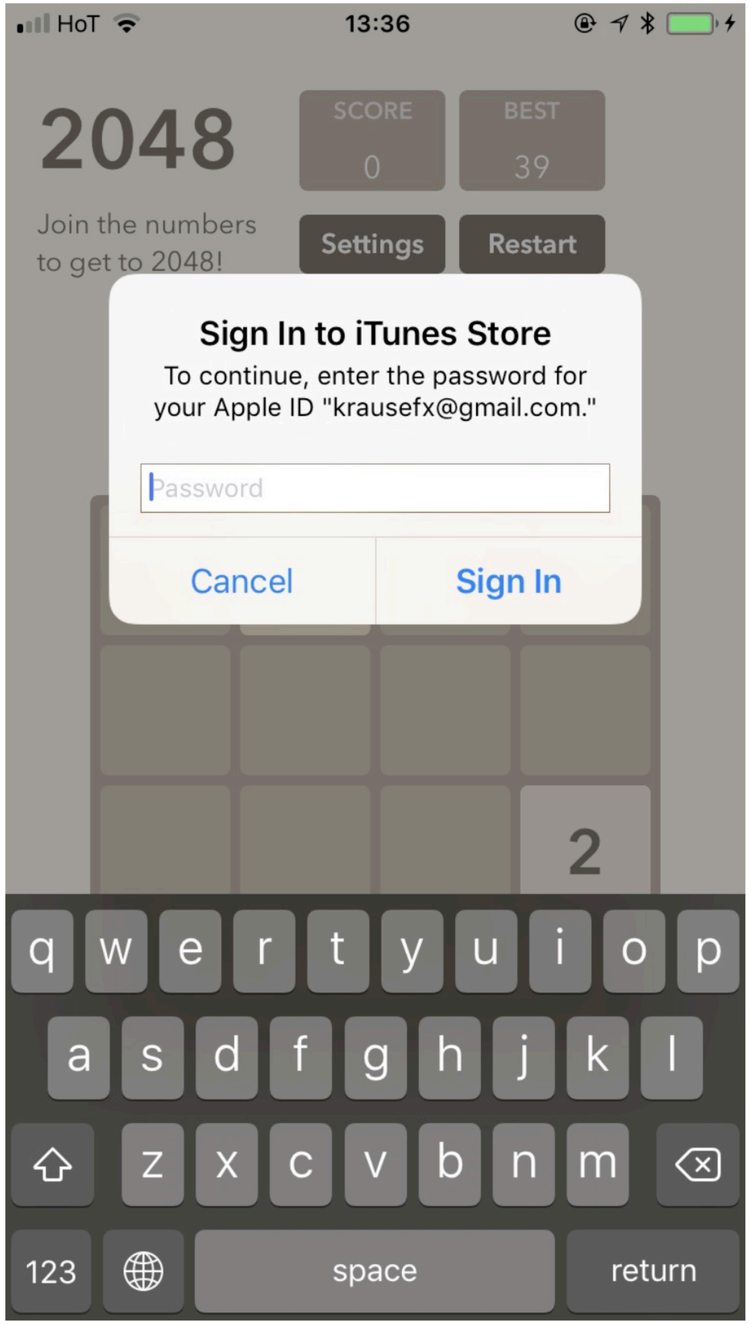

Search
Discov
Arcade
Create
Work
Play
Develo
Catego
Update
Aaron Parecki
Sign in to view account information.
If you have an Apple ID, sign in with it here. If you have used the iTunes Store or iCloud, for example, you have an Apple ID.
Apple ID: aaron@parecki.com
Password: required
Forgot Apple ID or Password?
Cancel
Sign In


icloud.com
Anyone can use Pages, Numbers, and Keynote for iCloud
Apple ID
Cancel
Enter the password for the user "Aaron Parecki" to continue.
Password:
Cancel
Continue
Continue with Password
Create Apple ID | System Status | Privacy Policy | Terms & Conditions | Copyright © 2021 Apple Inc. All rights reserved.


Keychain Access wants to use your confidential information stored in "iCloud" in your keychain.
To allow this, enter the "login" keychain password.
Password:
Always Allow
Deny
Allow


iCloud
icloud.com
Anyone can use Pages, Numbers, and Keynote for iCloud
Create your free Apple ID and get started today
Sign in to iCloud
aaron@parecki.com
Password
Keep me signed in
Forgot Apple ID or password?
Create Apple ID | System Status | Privacy Policy | Terms & Conditions | Copyright © 2021 Apple Inc. All rights reserved.





https://krausefx.com/blog/ios-privacy-stealpassword-easily-get-the-users-apple-id-password-just-by-asking

Google redirects
users to a
centralized account server

Secure | https://gmail.com

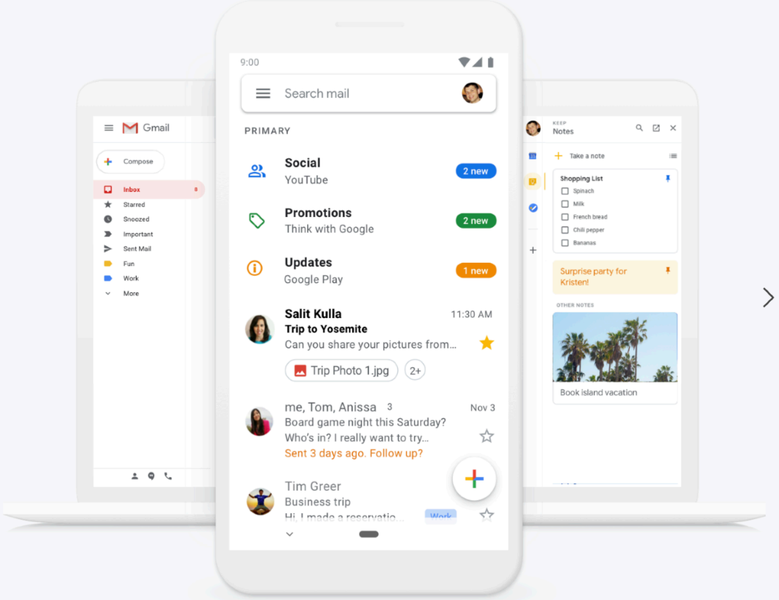
Gmail

For Work | Sign in | Create an account

Get more done with Gmail

Now more secure, smarter and easier to use—helping you save time and do more with your inbox.

Create an account



Secure | https://accounts.google.com/oauth/authorize?response_ty

Google

Sign in

to continue to Gmail

Email or phone

Forgot email?

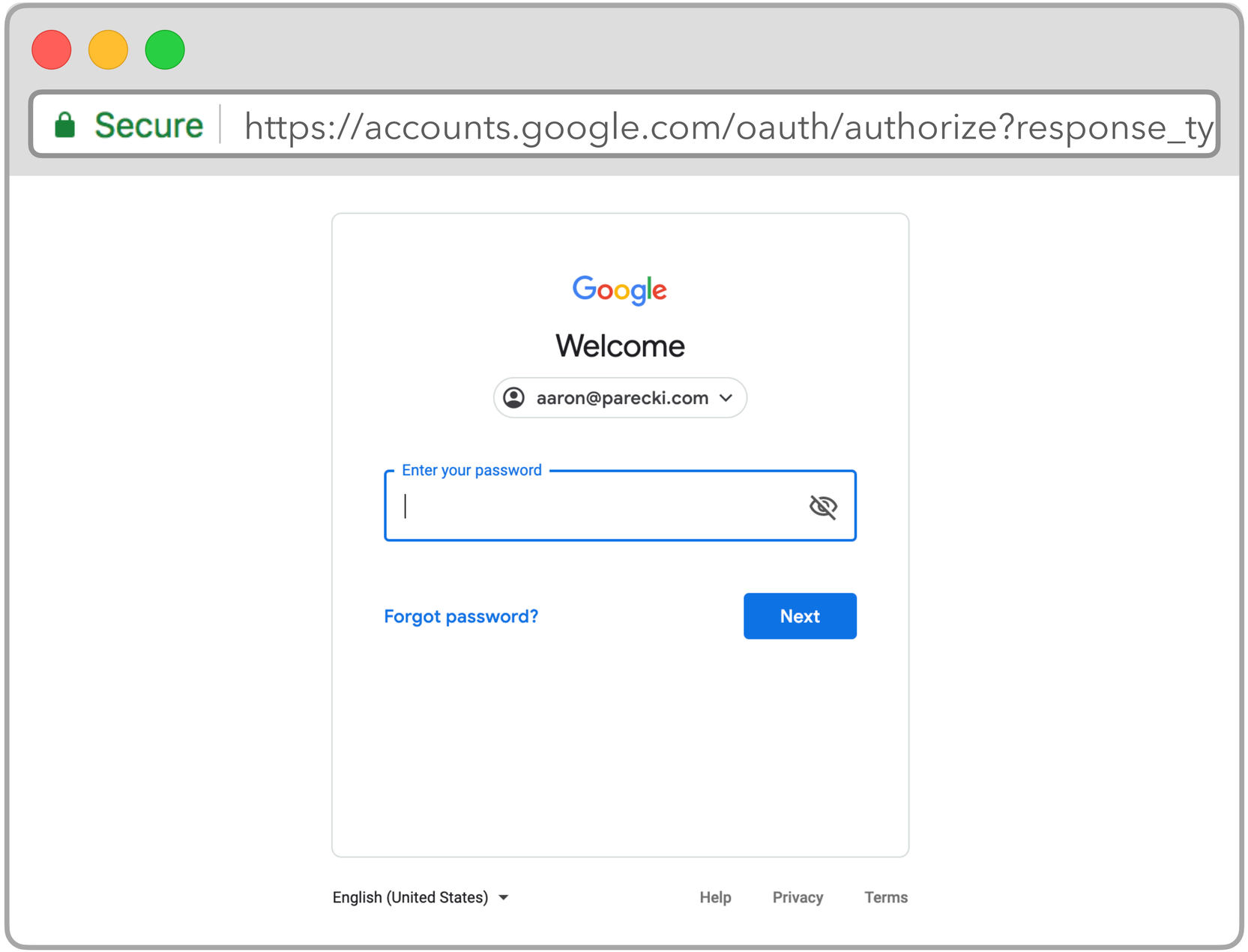
Not your computer? Use Guest mode to sign in privately.
Learn more

Create account

Next

English (United States)

Help Privacy Terms


Secure
https://accounts.google.com/oauth/authorize?response_ty
Google
Welcome
aaron@parecki.com
Enter your password
Forgot password?
Next
English (United States)
Help
Privacy
Terms

Secure | https://accounts.google.com/oauth/authorize?response_ty

Google

2-Step Verification

This extra step shows it's really you trying to sign in

aaron@parecki.com

Try another way to sign in

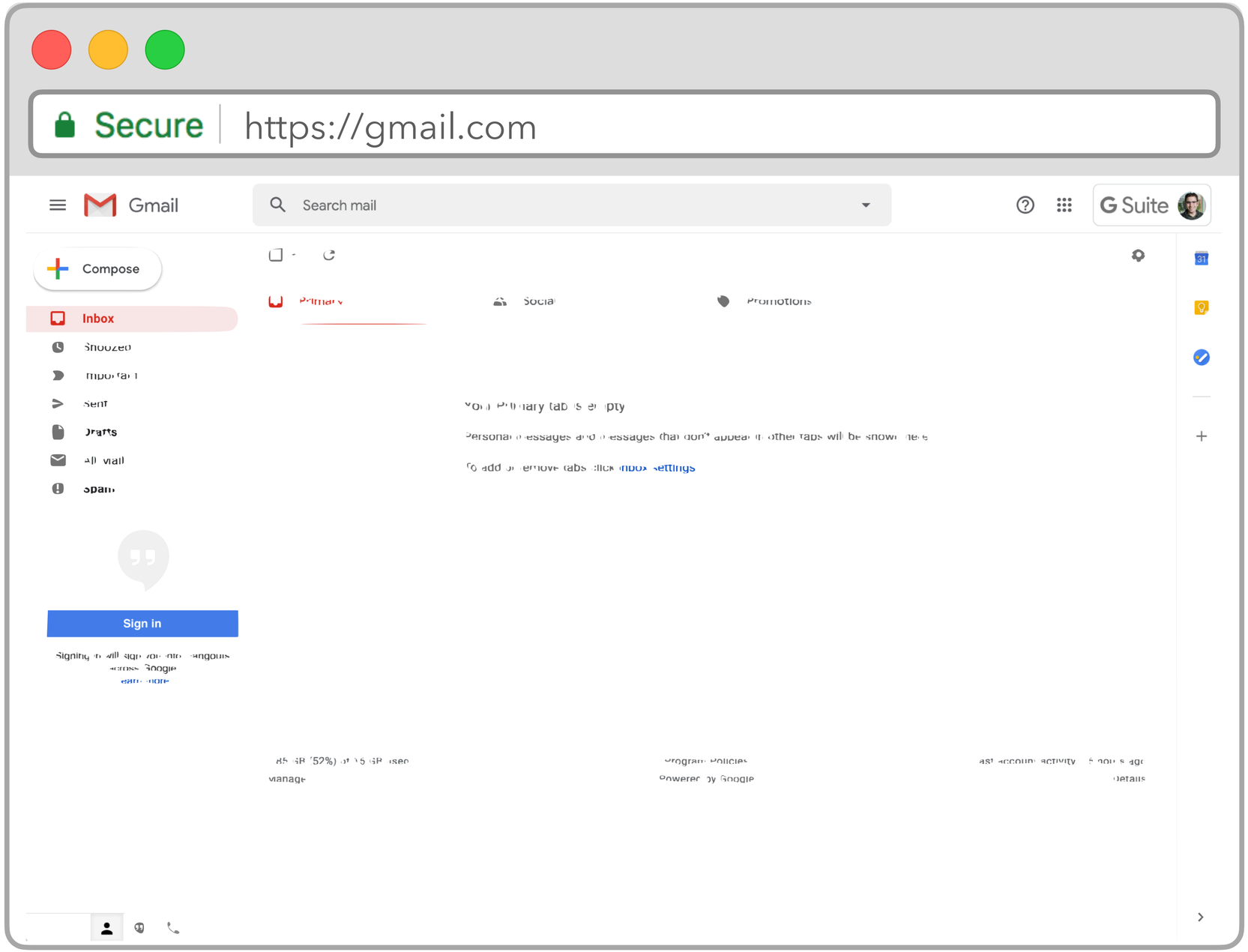
Use your Security Key

Get a one-time security code
Sign in on another device with your security key to get a code

Get a verification code from the **Google Authenticator** app

Get a verification code at (•••) •••-••42
Standard rates apply

Enter one of your 8-digit backup codes

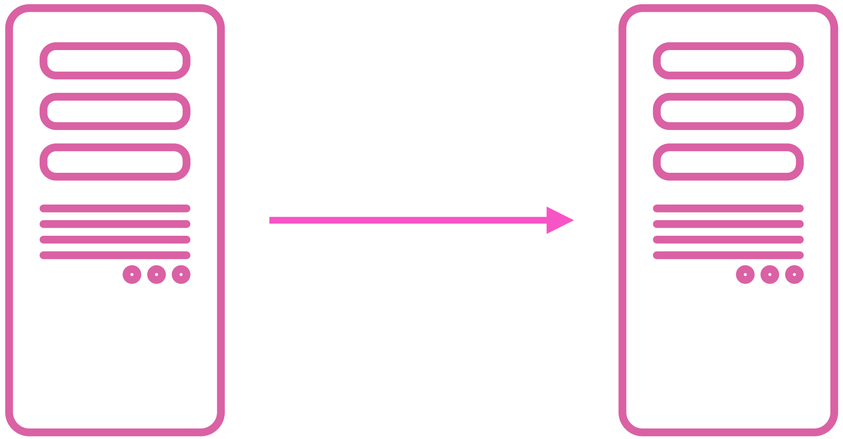

Secure
https://gmail.com
Gmail
Search mail
G Suite
Compose
Inbox
Sent
Drafts
Sign in
Primary
Social
Promotions

FRONT CHANNEL

VS

BACK CHANNEL

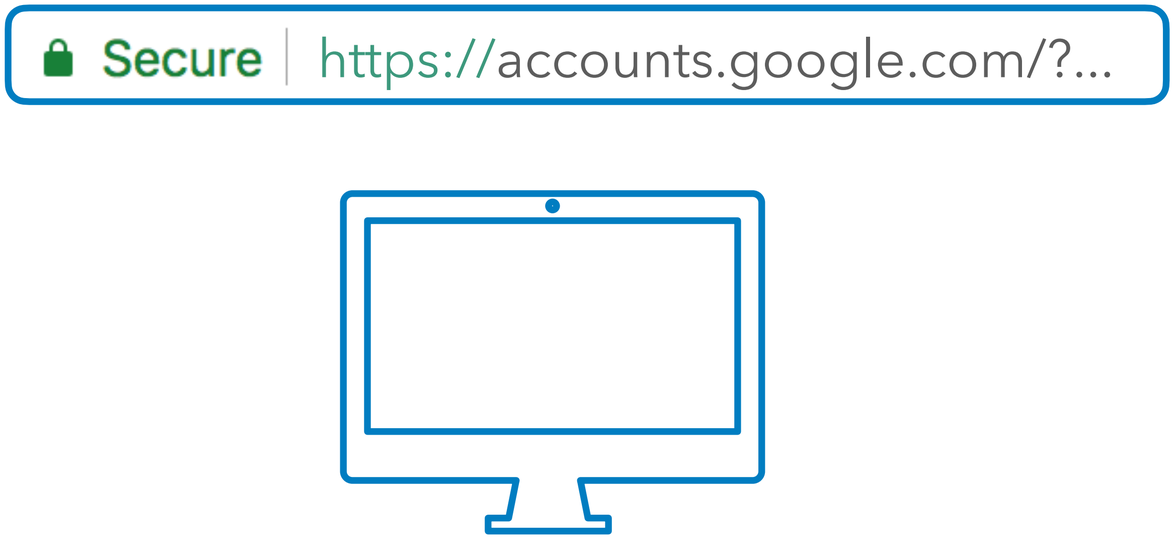
Back Channel



Sent from client to server

HTTPS request from client to server,
so requests cannot be tampered with

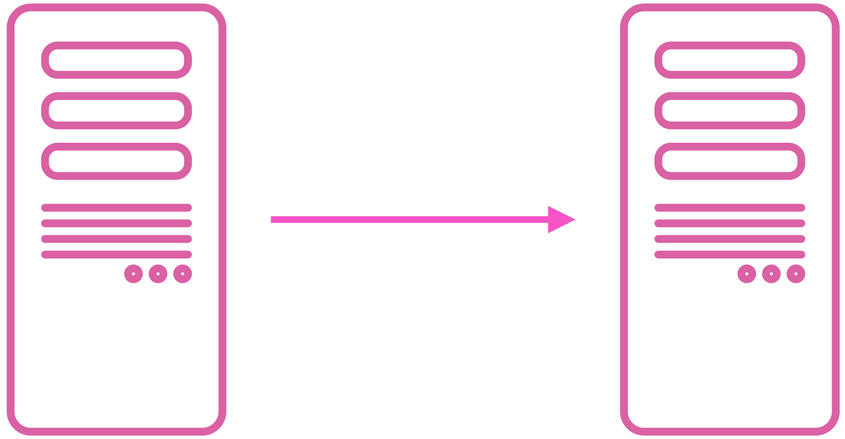
Front Channel



Passing data via the browser's address bar

The user, or malicious software,
can modify the requests and responses

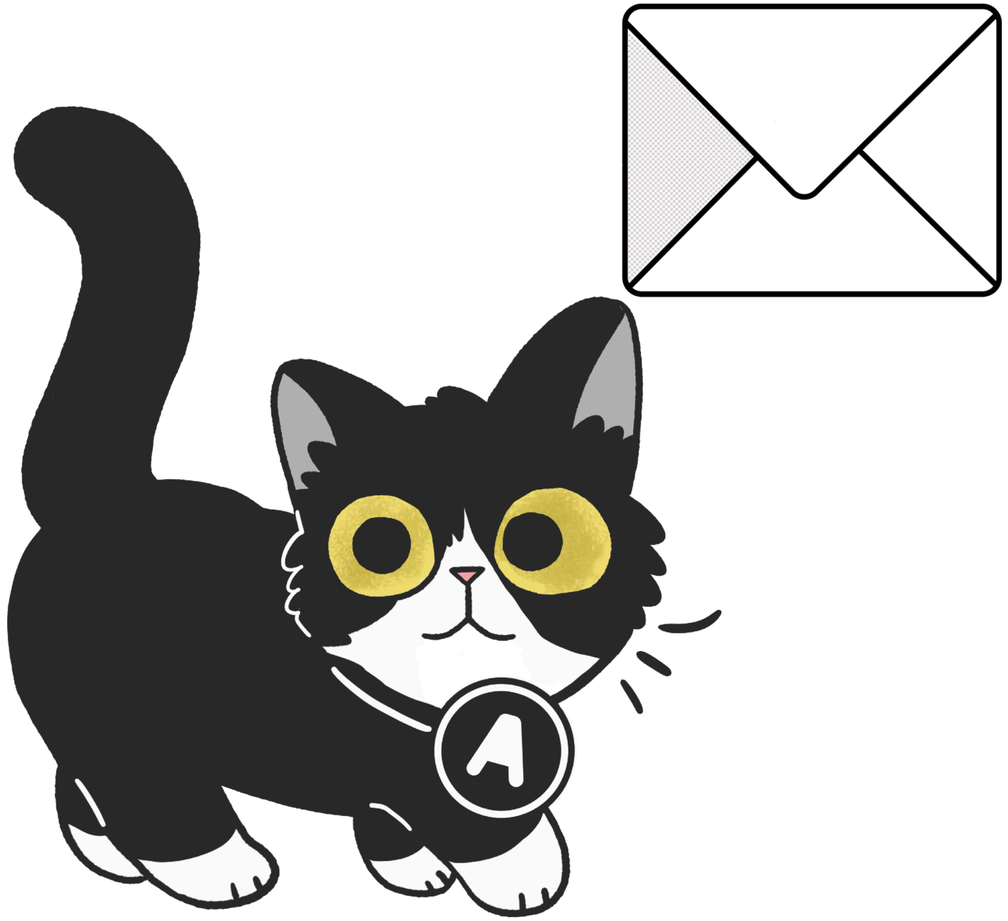
Back Channel Benefits



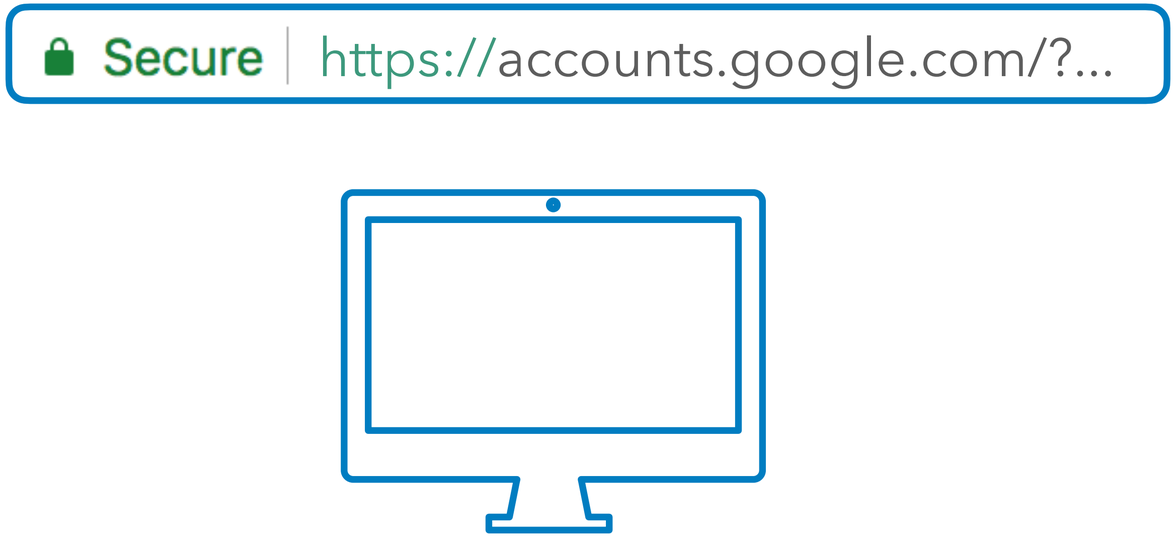
- The application knows it's talking to the right server
- Connection from app to server can't be tampered with
- Response from the server can be trusted because it came back in the same connection

Passing Data via the Back Channel

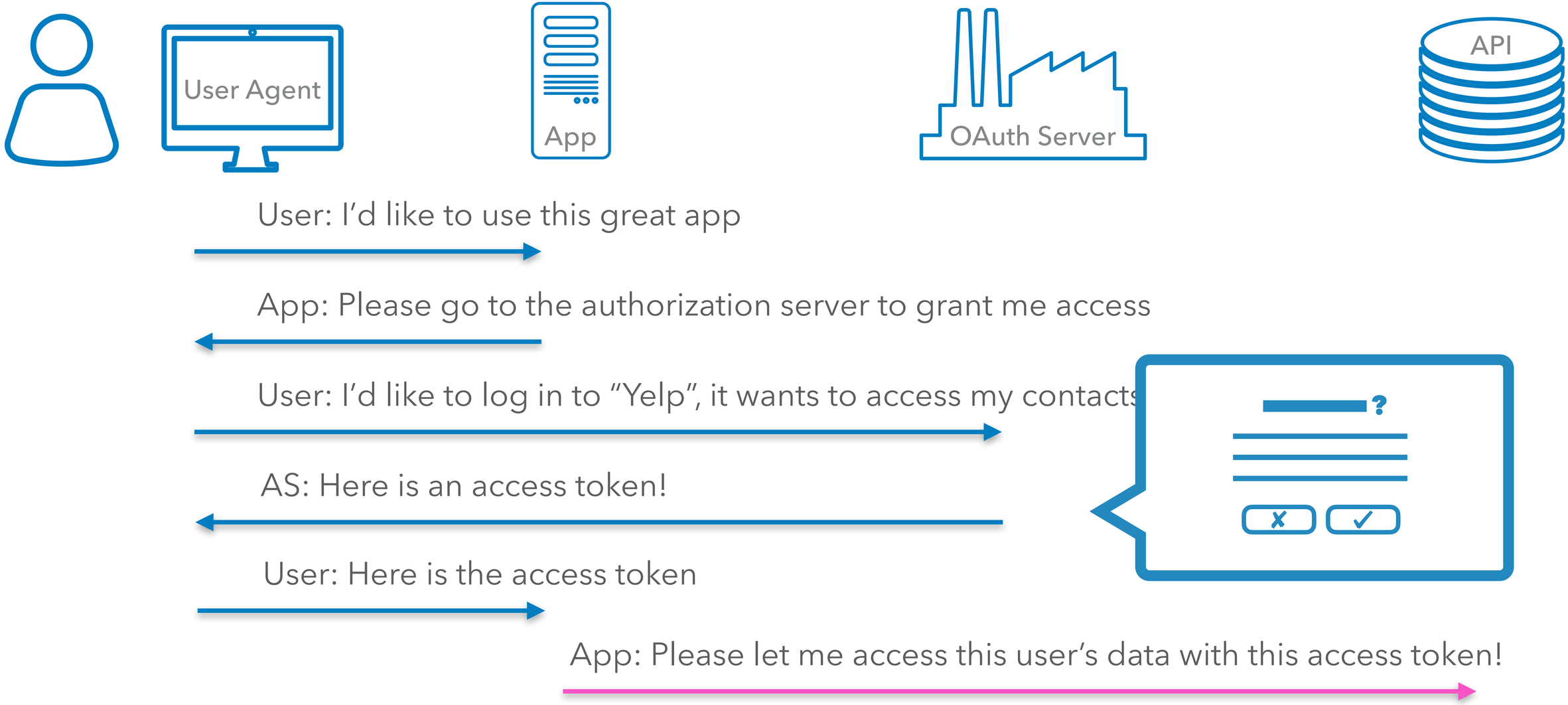
Passing Data via the Front Channel



Front Channel Benefits



- The user being involved enables them to give consent
- Enables easier two-factor authorization integration
- Doesn't require the receiver to have a publicly routable IP (e.g. can work on a phone)


User Agent
App
OAuth Server
API
User: I'd like to use this great app
App: Please go to the authorization server to grant me access
User: I'd like to log in to "Yelp", it wants to access my contacts
AS: Here is an access token!
User: Here is the access token
App: Please let me access this user's data with this access token!

https://tools.ietf.org/html/draft-ietf-oauth-security-topics

[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-lodderstedt-oauth-security-topics)
00 01 02 03 04 05 06 07 08 09 10 11
12 13

Web Authorization Protocol
Internet-Draft
Intended status: Best Current Practice
Expires: January 9, 2020

T. Lodderstedt
yes.com
J. Bradley
Yubico
A. Labunets
Facebook
D. Fett
yes.com
July 8, 2019

OAuth 2.0 Security Best Current Practice
draft-ietf-oauth-security-topics-13

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

QUIZ

GETTING READY:

CLIENT REGISTRATION

`client_id`

identifies the application

`client_secret`

the application's password

Create New Application

1 Platform — 2 Settings

An application in Okta represents an integration with the software you're building.
Choose your platform, and we'll recommend settings on the next step.

Native	Single-Page App	Web	Service
iOS, Android	Angular, React, etc.	.NET, Java, etc.	Machine-to-Machine

Previous | Cancel | Next

← Back to Applications

My Web App

Active View Logs

General Sign On Assignments

Client Credentials

Edit

Client ID

0oaeilg5ms1FUUljM0h7

Public identifier for the client that is required for all OAuth flows.

Client secret

••

Secret used by the client to exchange an authorization code for a token. This must be kept confidential! Do not include it in apps which cannot keep it secret, such as those running on a client.

Confidential Clients

Application running on a server

Has the ability to keep strings secret
since code is running in a trusted environment

Public Clients

The application can't keep strings secret

JavaScript/Single-Page apps: "view source"
Native apps: decompile and extract strings

EXERCISE 1

oauth.school

"Getting Started"

GRANT TYPE:

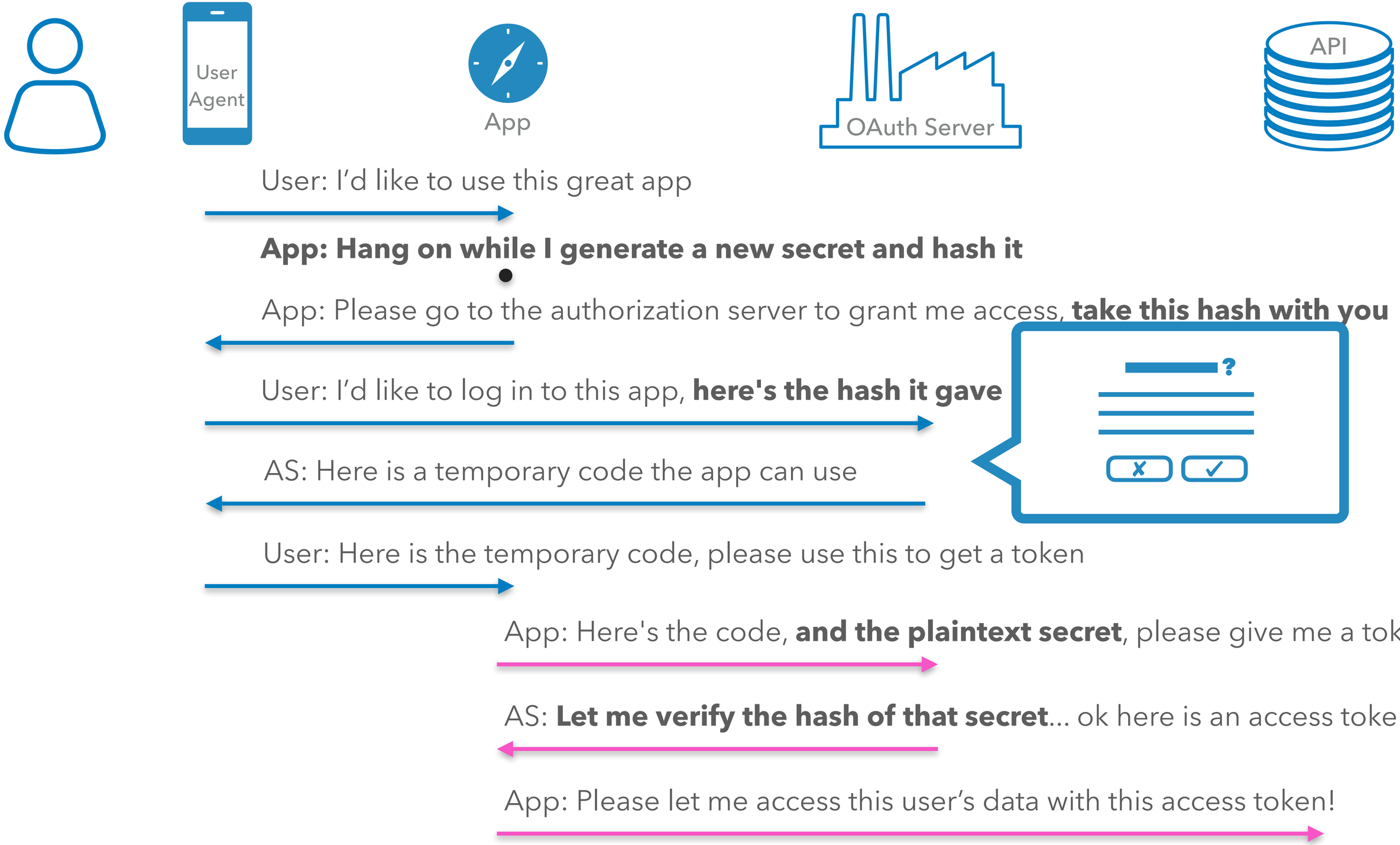
AUTHORIZATION CODE + PKCE

PKCE

(pronounced "pixie")

PROOF-KEY FOR CODE EXCHANGE

RFC 7636


User Agent
App
OAuth Server
API
User: I'd like to use this great app
App: Hang on while I generate a new secret and hash it
App: Please go to the authorization server to grant me access, take this hash with you
User: I'd like to log in to this app, here's the hash it gave
AS: Here is a temporary code the app can use
User: Here is the temporary code, please use this to get a token
App: Here's the code, and the plaintext secret, please give me a token
AS: Let me verify the hash of that secret... ok here is an access token!
App: Please let me access this user's data with this access token!

Generate the Code Verifier

Generate a random string 43-128 characters long

```
4A6hBupTkAtgbaQs39RSELUEqtSWTDTcRzVh1PpxD5YVKllU
```

Generate the Code Challenge

The challenge is the SHA256 hash of the verifier string

`base64url(sha256(code_verifier))`

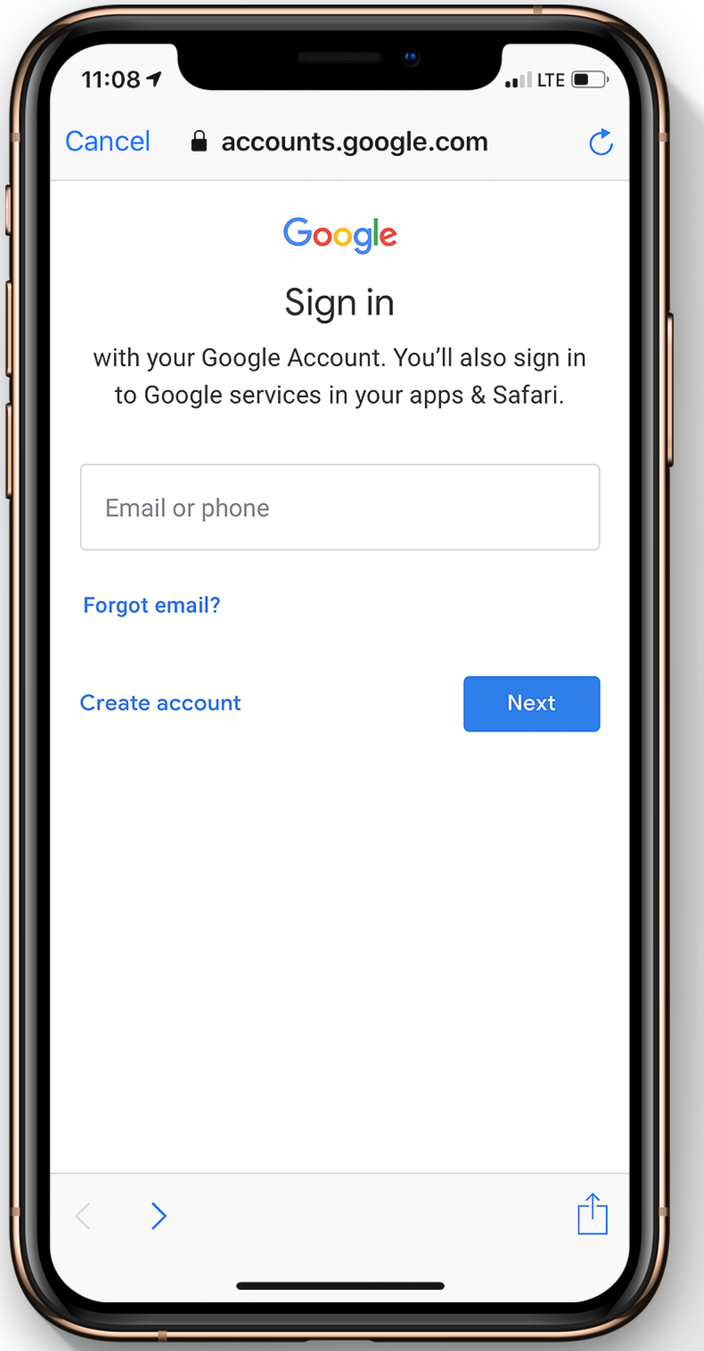
```
ipSBt30y481401NGbLjo026cqwsRQzR5KI40AuLAdZ8
```

The app builds the auth link

Front Channel

Include the code challenge (the hashed value) in the request

```
https://authorization-server.com/auth?
  response_type=code&
  client_id=CLIENT_ID&
  redirect_uri=REDIRECT_URI&
  scope=photos&
  state=1234zyx&
  code_challenge=XXXXXXXXXXXXXX&
  code_challenge_method=S256
```



Front Channel

Front Channel

If User Denies

The user is redirected back to the application with an error code

```
example://callback?error=access_denied&state=1234xyz
```

If User Allows

The user is redirected back to the application with an authorization code

```
example://callback?
  code=AUTH_CODE_HERE&
  state=1234zyx
```

Exchange the Code for an Access Token

Back Channel

Verify state, then make a POST request:

- **grant_type=authorization_code** - indicates that this request contains an authorization code
- **code=CODE_FROM_QUERY** - Include the authorization code from the query string of this request
- **redirect_uri=REDIRECT_URI** - This must match the `redirect_uri` used in the original request
- **client_id=CLIENT_ID** - The client ID you received when you first created the application
- **code_verifier=VERIFIER_STRING** - The *plaintext* code verifier initially created

Exchange the Code for an Access Token

Back Channel

```
POST https://api.authorization-server.com/token
  grant_type=authorization_code&
  code=AUTH_CODE_HERE&
  redirect_uri=REDIRECT_URI&
  client_id=CLIENT_ID&
  code_verifier=VERIFIER_STRING
```

Note: code verifier used in place of client secret

Exchange the Code for an Access Token

Back Channel

The server compares the **`code_verifier`** with the **`code_challenge`** that was in the request when it generated the authorization code, and responds with an access token.

```
{
  "access_token":"RsT5OjbzRn430zqMLgV3Ia",
  "expires_in":3600,
  "refresh_token":"64d049f8b2119a12522d5dd96d5641af5e8"
}
```

PKCE: Proof Key for Code Exchange

- Authenticates the code exchange step
- Does not authenticate the app itself
- Safe for clients to do with authorization servers that don't support PKCE
- Recent update to the OAuth security recommendations encouraging PKCE even for confidential clients

AppAuth.io

iOS / Android / JavaScript

EXERCISE 2

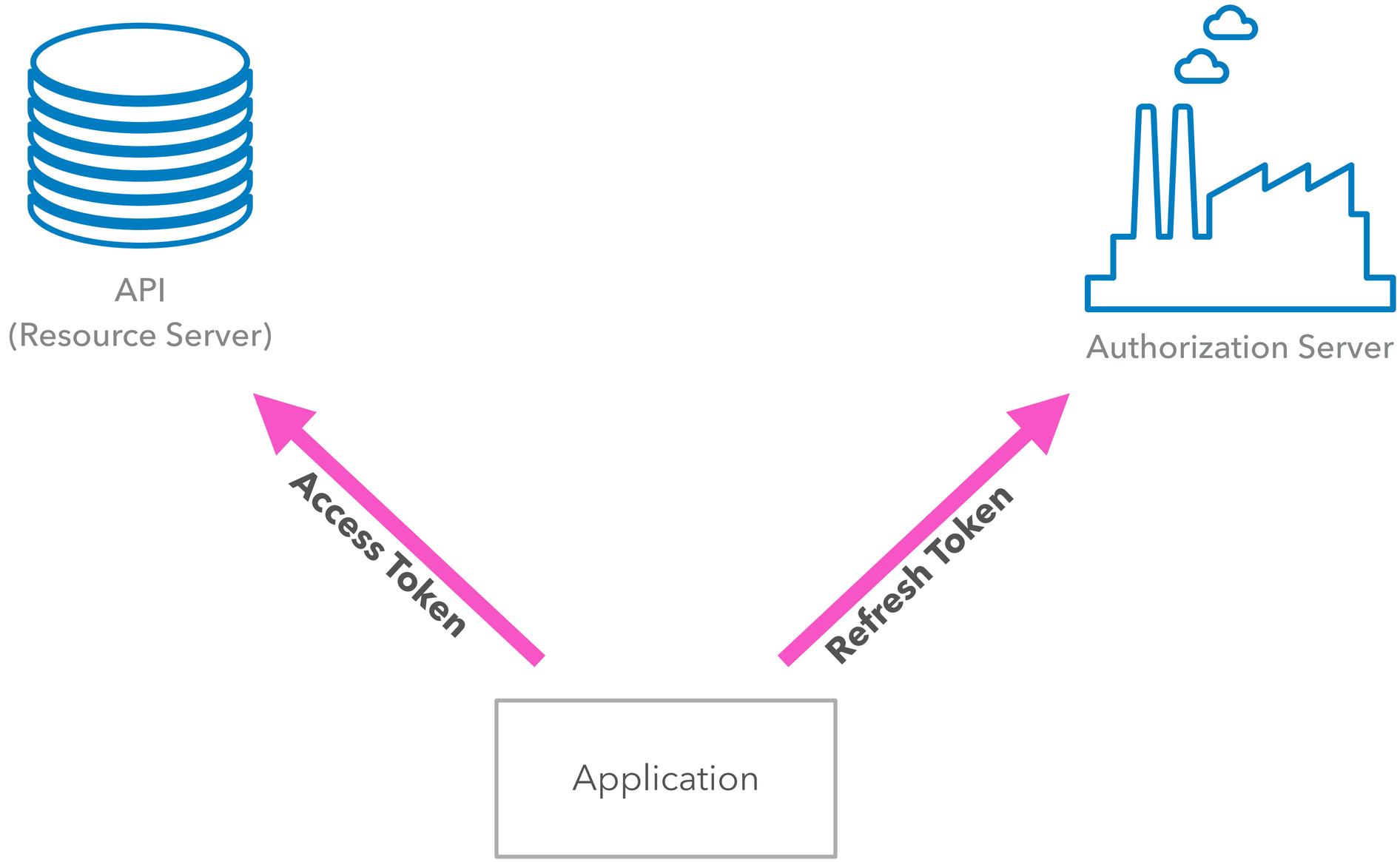
oauth.school

"OAuth for Web Applications"

REFRESH TOKENS

WHAT IS A REFRESH TOKEN?

- A special token used to get new access tokens
- Keeps the user logged in without having to visit the authorization server
- Sometimes requested along with the access token in the initial step
- Usually requires scope: `offline_access`
- Usually not issued to JavaScript apps

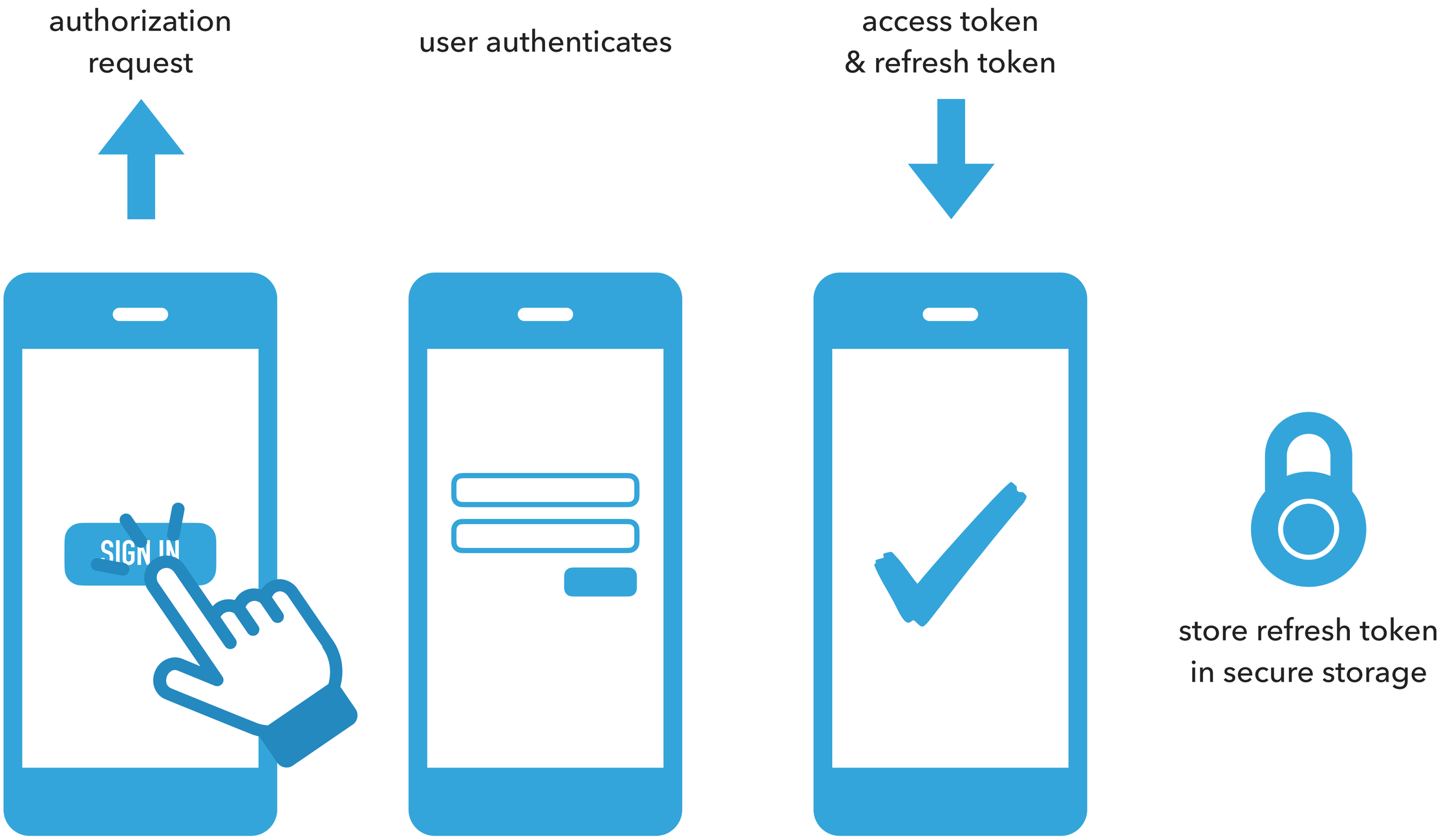

API
(Resource Server)
Authorization Server
Access Token
Refresh Token
Application

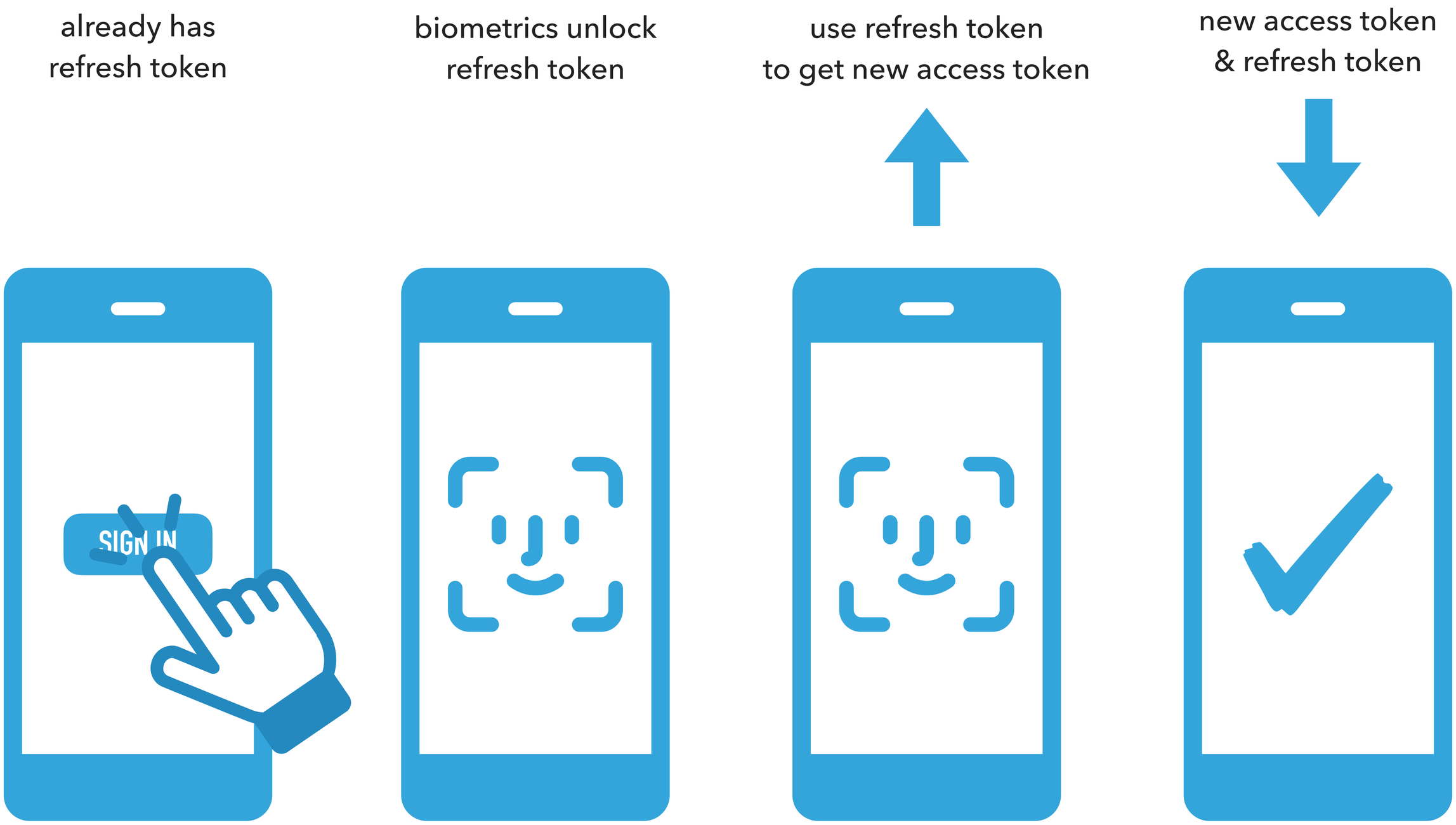
Exchange the Refresh Token for an Access Token

```
POST https://authorization-server.com/token
  grant_type=refresh_token&
  refresh_token=REFRESH_TOKEN&
  client_id=CLIENT_ID&
  client_secret=CLIENT_SECRET
```

New Access Token in the Response

```
{
  "access_token": "RsT5OjbzRn430zqMLgV3Ia",
  "expires_in": 3600,
  "refresh_token": "64d049f8b21191e12522d5d96d5641af5e8"
}
```


authorization
request
user authenticates
access token
& refresh token
SIGN IN
store refresh token
in secure storage


already has
refresh token
biometrics unlock
refresh token
use refresh token
to get new access token
new access token
& refresh token
SIGN IN

EXERCISE 3

oauth.school

"Refresh Tokens"

OPENID CONNECT:

WHO LOGGED IN?

Gives the application a way to make API requests

Accessing APIs

Tells the application about the user (ID, name, email, etc)

Identification

OpenID Connect

OAUTH 2

New token type: ID Token

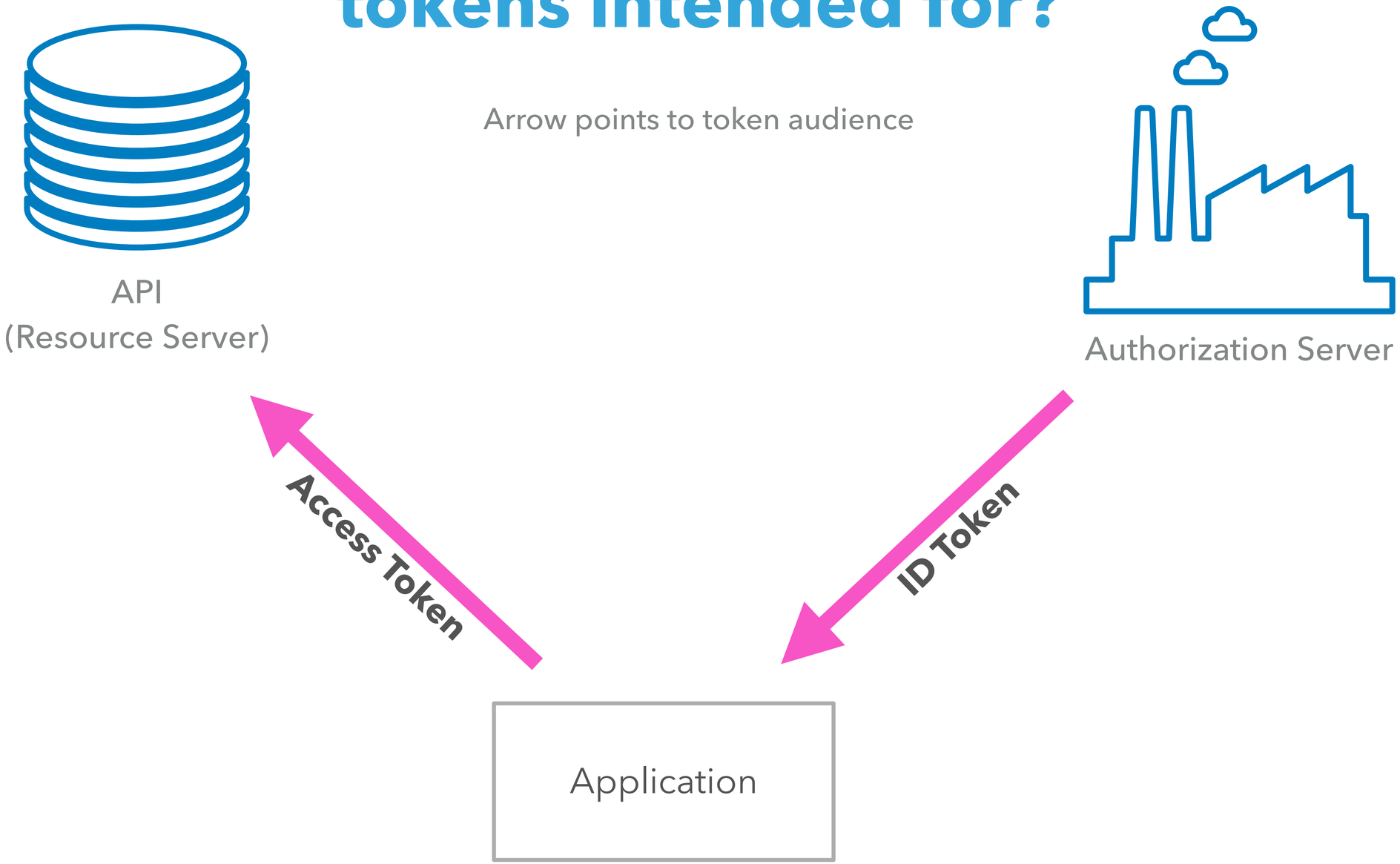
An ID token encodes user information

eyJraWQiOiJiRmxZbmkzLXRhMXFSa0lFellHc2tLeFFRVUJvczZnOU9RQnRmNm9xcUxJIiwiYW
xnIjoiUlMyNTYifQ.eyJzdWIiOiIwMHVjcTNid2o0V25JcTNnejBoNyIsIm5hbWUiOiJQYWRtY
S0yIEdvdmluZGFyYWphbHUiLC
6Ly9wYWRtYWdvdm
XVkIjoiMG9hZD
1OTg1MTAsImp
lJ6dkEiLCJhb
iOiJhYmMiLC
29tIiwiZ2l
5X25hbWUiO
iwidXBkYX
dCf1BWZfH
HjVI8ALQNwtM7FnIs9H6gCH0
Al_y9z5mvLyDynf9IjrOyTPo
t0_6PKG7sSkhbMeBicIc35SJJaXt81KSfkYDUp5s1UQ74ATHrtLe7HMU1yp_KajgYUKxMXO5Ni
XpeNEHzarAOWzLHblrQcgkpuJbY3KM1HHg

This may look like an access token if your access tokens are also JWTs, but they are not the same!

Who are the different tokens intended for?

Arrow points to token audience



ID Token: JWT

header


```
eyJraWQiOiJiRmxZbmkzLXRhMXFSa0lFellHc2tLeFFRVUJvczZnOU9RQnRmNm9xcUxJIiwiYWxnI
joiUlMyNTYifQ
.
eyJzdWIiOiIwMHVjcTNid2o0V25JcTNnejBoNyIsIm5hbWUiOiJQYWRtYS0yIEdvdmluZGFyYWphb
HUiLCJsb2NhbGUiOiJlbi1VUyIsInZlciI6MSwiaXNzIjoiaHR0cHM6Ly9wYWRtYWdvdmluZGFyYW
phbHUub2t0YXByZXZpZXcuY29tL29hdXRoMi9kZWZhdWx0IiwiYXVkIjoiMG9hZDlydTd0endmNUF
qcGIwaDcgIiwiaWF0IjoxNTI0NTk0OTEwLCJleHAiOjE1MjQ1OTg1MTAsImp0aSI6IklELklfNUc4
RzhWdXowMHJvYl9aSzlja3J0T0pseVdwNzhxMU5naGV2QlJ6dkEiLCJhbXIiOlsicHdkIl0sImlkc
CI6IjAwb2NxM2J3aTFoTnpRT3B5MGg3Iiwibm9uY2UiOiJhYmMiLCJwcmVmZXJyZWRfdXNlcm5hbW
UiOiJwYWRtYS5nb3ZpbmRhcmFqYWx1QG9rdGEuY29tIiwiZ2l2ZW5fbmFtZSI6IlBhZG1hIiwibWl
kZGxlX25hbWUiOiJLcmlzaG5hIiwiZmFtaWx5X25hbWUiOiJHb3ZpbmRhcmFqYWx1Iiwiem9uZWlu
Zm8iOiJBbWVyaWNhL0xvc19BbmdlbGVzIiwidXBkYXRlZF9hdCI6MTUyNDU5NDM2MSwiYXV0aF90a
W1lIjoxNTI0NTk0OTA3fQ
.
HvMYW8XbdCf1BW-
ZfHQ1odaAYJjZqKkh1NUkHW0clk6J7pYunn8jllbIp0IhSjcCn6PBIlZPrrE0dkuyjvdHjVI8ALQN
wtM7FnIs9H6gCH0oONx4EL4K-Ef4d_w46qeqsCwMClvNoaE3c2I5-kON-
uJUlaefbnr6Al_y9z5mvLyDynf9IjrOyTPoIrgk9V46l28Aulp4dJhqBtZfpYyVbKrXawHSO5FvKT
DMPBhQgxt0_6PKG7sSkhbMeBicIc35SJJaXt81KSfkYDUp5s1UQ74ATHrtLe7HMU1yp_KajgYUKxM
XO5NiXpeNEHzarAOWzLHblrQcgkpuJbY3KM1HHg
```


payload

signature

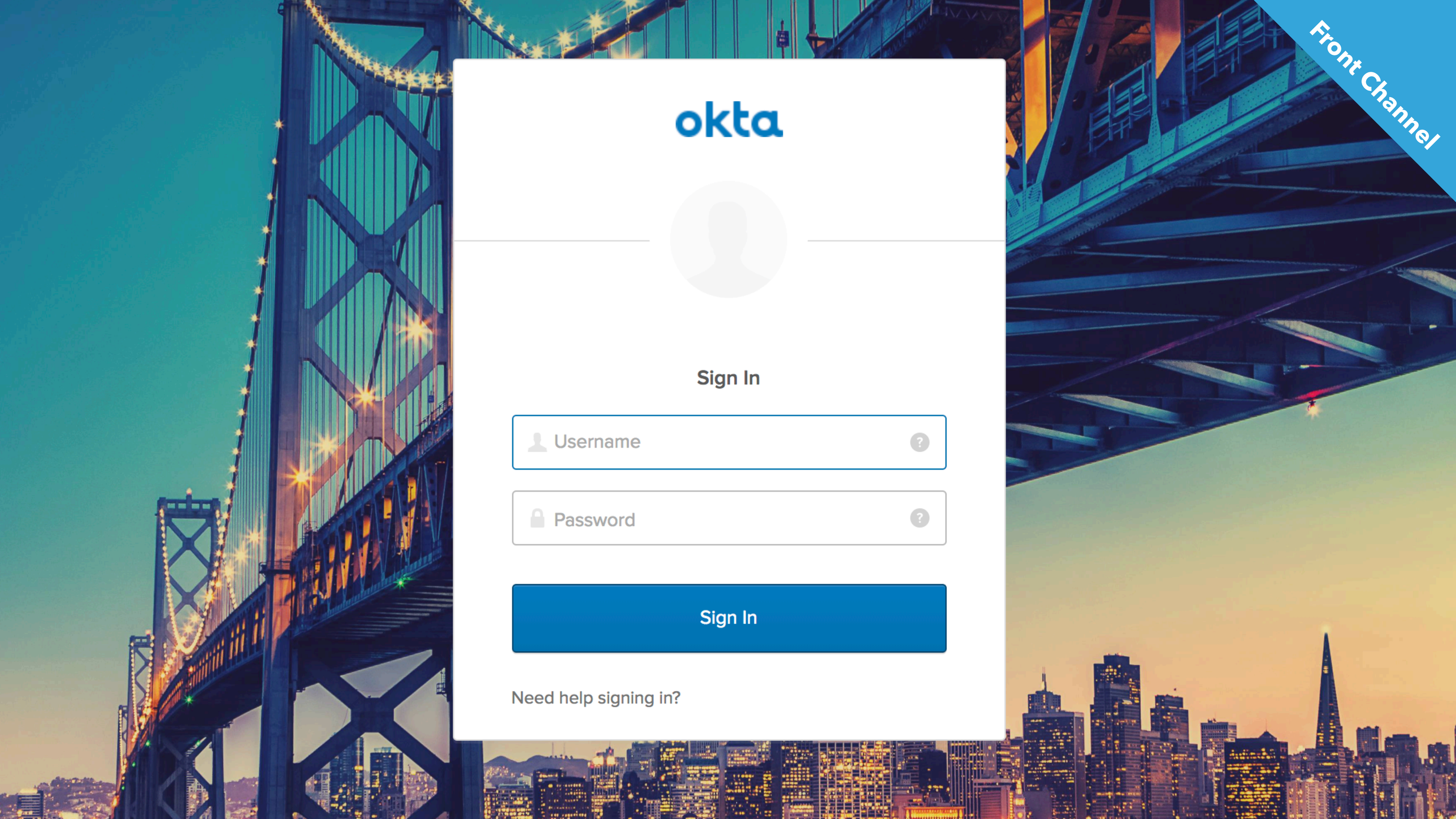
Decoded ID Token

```
header
.
{
  "sub": "{USER_ID}",
  "iss": "https://authorization-server.com/oauth2/ausd1nry9hBoyvKrY0h7",
  "aud": "{CLIENT_ID}",
  "iat": 1524237221,
  "exp": 1524240821,
  "nonce": "{NONCE}",
  "auth_time": 1524606562
}
.
signature
```

OpenID Connect - Authorization Code Flow + PKCE

Front Channel

```
https://authorization-server.com/auth?
  response_type=code&
  client_id=CLIENT_ID&
  redirect_uri=REDIRECT_URI&
  scope=openid+profile&
  state=xyz1234&
  nonce=1029385476&
  code_challenge=CODE_CHALLENGE&
  code_challenge_method=S256
```


Front Channel
okta
Sign In
Username
Password
Sign In
Need help signing in?

If User Allows Access

Front Channel

The user is redirected back to the redirect_uri with an authorization code

```
https://example-app.com/redirect?
code=RhcmFqYWx1Iiwiem9uZWluZm8iOiJBbWVyaWNhL0xvc&state=xyz1234
```

Exchange the Code for an ID Token

Back Channel

```
POST https://api.authorization-server.com/token
  grant_type=authorization_code&
  code=AUTH_CODE_HERE&
  redirect_uri=REDIRECT_URI&
  client_id=CLIENT_ID&
  code_verifier=VERIFIER_STRING
```

Exchange the Code for an ID Token

Back Channel

The server returns an ID token from the token endpoint, so the client can skip the JWT verification step since the verification has already happened thanks to PKCE and HTTPS.


```
{
  "access_token":"RsT5OjbzRn430zqMLgV3Ia",
  "expires_in":3600,
  "id_token":"eyJraWQiOiJiRmxZbmkzLXRhMXFSa0lFellHc2..."
}
```

An ID Token


```
eyJraWQiOiJiRmxZbmkzLXRhMXFSa0lFellHc2tLeFFRVUJvczZnOU9RQnRmNm9xcUxJIiwiYW
xnIjoiUlMyNTYifQ.eyJzdWIiOiIwMHVjcTNid2o0V25JcTNnejBoNyIsIm5hbWUiOiJQYWRtY
S0yIEdvdmluZGFyYWphbHUiLCJsb2NhbGUiOiJlbi1VUyIsInZlciI6MSwiaXNzIjoiaHR0cHM
6Ly9wYWRtYWdvdmluZGFyYWphbHUub2t0YXByZXZpZXcuY29tL29hdXRoMi9kZWZhdWx0IiwiY
XVkIjoiMG9hZDlydTd0endmNUFqcGIwaDcgIiwiaWF0IjoxNTI0NTk0OTEwLCJleHAiOjE1MjQ
1OTg1MTAsImp0aSI6IklELklfNUc4RzhWdXowMHJvYl9aSzlja3J0T0pseVdwNzhxMU5naGV2Q
lJ6dkEiLCJhbXIiOlsicHdkIl0sImlkcCI6IjAwb2NxM2J3aTFoTnpRT3B5MGg3Iiwibm9uY2U
iOiJhYmMiLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJwYWRtYS5nb3ZpbmRhcmFqYWx1QG9rdGEuY
29tIiwiZ2l2ZW5fbmFtZSI6IlBhZG1hIiwibWlkZGxlX25hbWUiOiJLcmlzaG5hIiwiZmFtaWx
5X25hbWUiOiJHb3ZpbmRhcmFqYWx1Iiwiem9uZWluZm8iOiJBbWVyaWNhL0xvc19BbmdlbGVzI
iwidXBkYXRlZF9hdCI6MTUyNDU5NDM2MSwiYXV0aF90aW1lIjoxNTI0NTk0OTA3fQ.HvMYW8Xb
dCf1BWZfHQ1odaAYJjZqKkh1NUkHW0clk6J7pYunn8jllbIp0IhSjcCn6PBIlZPrrE0dkuyjvd
HjVI8ALQNwtM7FnIs9H6gCH0oONx4EL4KEf4d_w46qeqsCwMClvNoaE3c2I5kONuJUlaefbnr6
Al_y9z5mvLyDynf9IjrOyTPoIrgk9V46l28AuIp4dJhqBtZfpYyVbKrXawHSO5FvKTDMPBhQgx
t0_6PKG7sSkhbMeBicIc35SJJaXt81KSfkYDUp5s1UQ74ATHrtLe7HMU1yp_KajgYUKxMXO5Ni
XpeNEHzarAOWzLHblrQcgkpuJbY3KM1HHg
```

Decoded ID Token

```
header
.
{
  "sub": "{USER_ID}",
  "email": "user@example.com",
  "name": "User Name",
  "iss": "https://authorization-server.com/oauth2/ausd1nry9hBoyvKrY0h7",
  "aud": "{CLIENT_ID}",
  "iat": 1524237221,
  "exp": 1524240821,
  "nonce": "{NONCE}",
  "auth_time": 1524606562
}
.
signature
```

Verifying the JWT

- If the ID token was received via the front channel, the data can't be trusted until it's verified
- Fetch the public key of the server that issued the token
- Verify the signature matches

Validating the Claims

- **issuer (iss)** – does the token originate from the right auth server?
- **audience (aud)** – is the token intended for my application?
- **expiry time (exp)** – has the current time already passed the exp time?
- **issued at time (iat)** – is the current time too far from the issued time?
- **nonce** - does it match the nonce passed in the request?

EXERCISE 4

oauth.school

"OpenID Connect"

RESOURCES

Sponsored By
okta

OAuth 2.0 Servers

Written by Aaron Parecki

OAuth 2.0 is the modern standard for securing access to APIs.

Read on for a complete guide to building your own authorization server.

Learn about OAuth 2.0

Solve it with Okta

Table of Contents

Search for a Topic

oauth.com

Sponsored By okta

Return to OAuth Docs →

OAuth 2.0 Playground

The OAuth 2.0 Playground will help you understand the OAuth authorization flows and show each step of the process of obtaining an access token.

Choose an OAuth flow

To begin, register a client and a user (don't worry, we'll make it quick)

Authorization Code | PKCE | Implicit | Device Code | OpenID Connect

oauth.com/playground

oauth.net

OAuth 2.0 Code Articles Security Books About

An **open protocol** to allow **secure authorization** in a **simple** and **standard** method from web, mobile and desktop applications.

Learn more about OAuth 2.0 »

The OAuth 2.0 authorization framework enables third-party applications to obtain limited access to a web service.

For Consumer developers...

If you're building...

- web applications
- desktop applications
- mobile applications
- Javascript or browser-based apps

OAuth is a simple way to publish and interact with protected data. It's also a safer and more secure way for people to give you access. We've kept it simple to save you time.

For Service Provider developers...

If you're supporting...

- web applications
- mobile applications
- server-side APIs
- mashups

If you're storing protected data on your users' behalf, they shouldn't be spreading their passwords around the web to get access to it. Use OAuth to give your users access to their data while protecting their account credentials.

Get started...

April 10, 2018

What is the OAuth 2.0 Authorization Code Grant Type?

Aaron Parecki

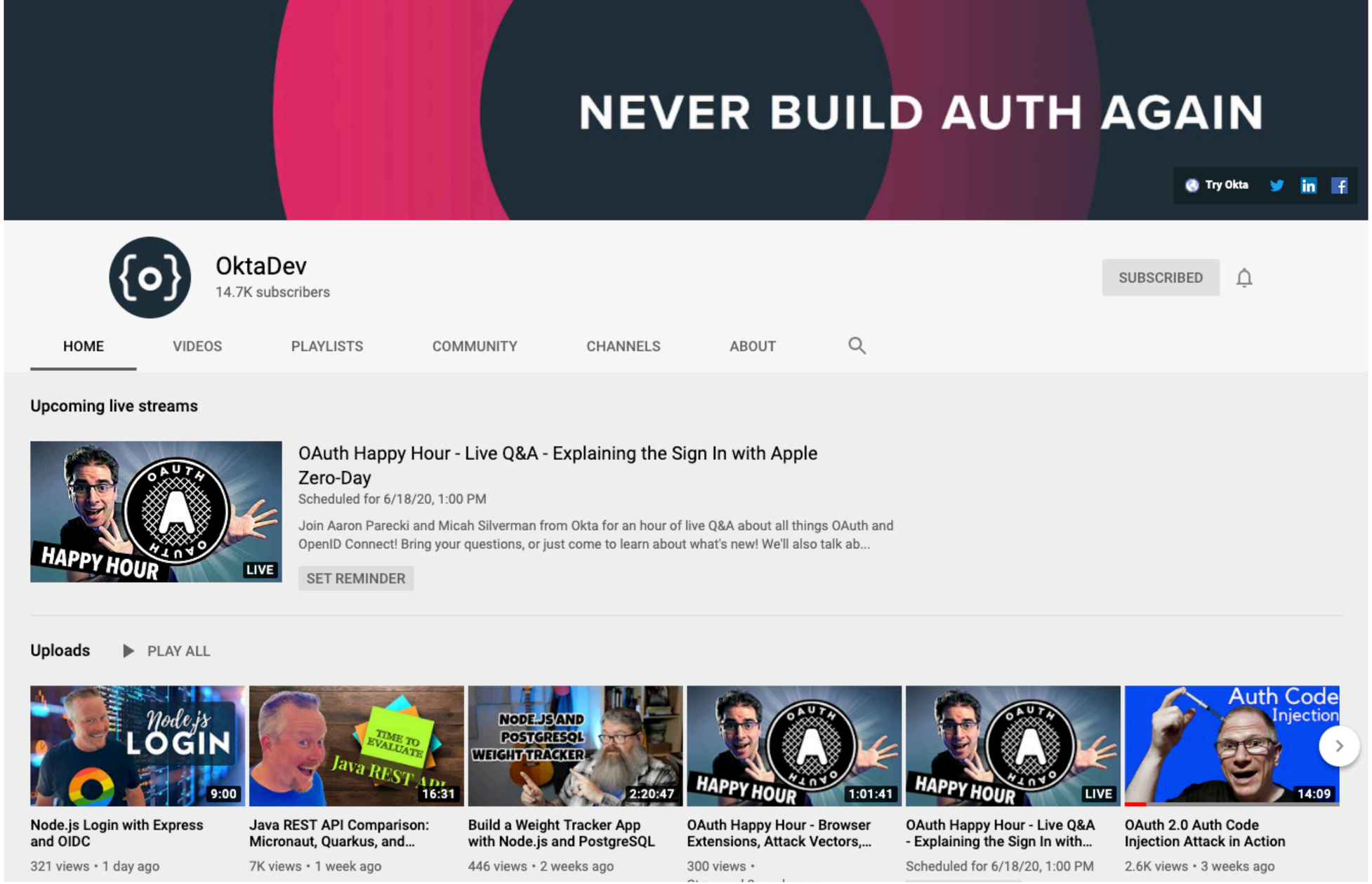
The Authorization Code Grant Type is probably the most common of the OAuth 2.0 grant types that you'll encounter. It is used by both web apps and native apps to get an access token after a user authorizes an app.

This post is the first part of a series where we explore frequently used OAuth 2.0 grant types. If you want to back up a bit and learn more about OAuth 2.0 before we dive in, check out What the Heck is OAuth?, also on the Okta developer blog.

What is an OAuth 2.0 Grant Type?

In OAuth 2.0, the term "grant type" refers to the way an application gets an access token. OAuth 2.0 defines several grant types, including the authorization code flow. OAuth 2.0 extensions can also define new grant types.

developer.okta.com/blog



youtube.com/oktadev

{okta} PRODUCT PRICING BLOG DOCS SUPPORT LOGIN SIGN UP

Add Auth in minutes (with no credit card)

$0 per month

Up to 1,000 monthly active users

GET STARTED FREE

Identity and access control for your app

- Authentication
- Authorization
- Basic multi-factor authentication
- User management
- Email support
- Customizable sign-in and registration

Products

Developer

Priced at

$0

per month for up to

1,000 MAUs

START FREE

One App

High availability, premiere support and additional MAUs for scaling your application.

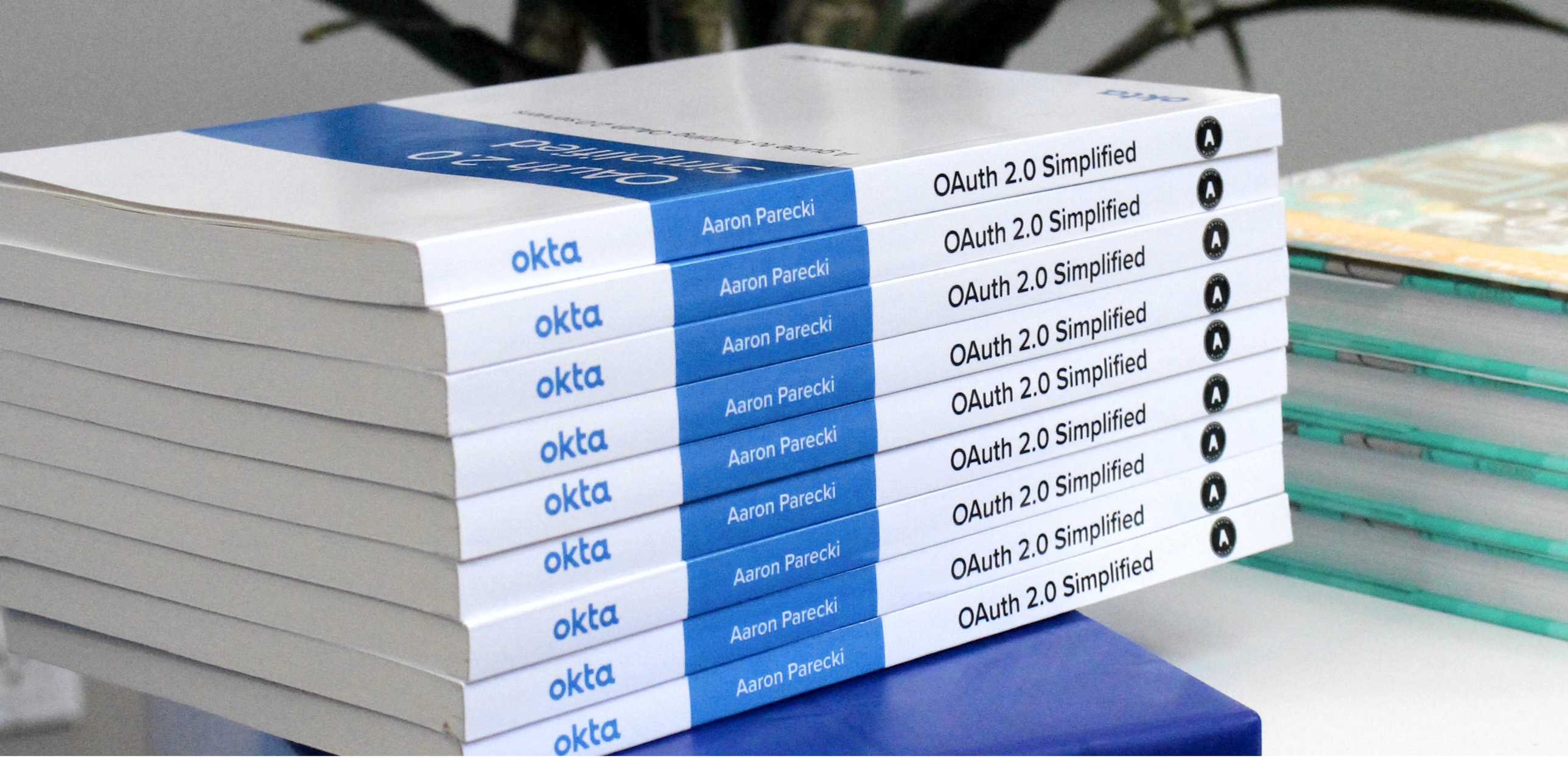
CONTACT US

Enterprise

Reliable authentication and authorization for complex environments with multiple applications.

CONTACT US

developer.okta.com/signup



oauth2simplified.com

Thank You!

@aaronpk

oauth2simplified.com